TURBO DEBUGGER®

BENUTZER-
HANDBUCH

BORLAND

Turbo Debugger

Turbo Debugger 3.0

Benutzerhandbuch

B O R L A N D

Copyright © 1991 by Borland International, INC.
Jede Vervielfältigung dieses Handbuches sowie des ObjectWindows Softwareprogrammes wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und die Rechte an dem Softwareprogramm ObjectWindows liegen bei Borland International.

Der rechtmäßige Erwerb der Programmdisketten und der Handbücher erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm ObjectWindows nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.

Sie (der Lizenznehmer von ObjectWindows) dürfen eigene Programme, die Sie mit dem Borland C++-Sprachcompiler geschrieben und compiliert haben, ohne Einschränkung oder Verpfichtung zu zusätzlichen Lizenzgebühren verwenden, weitergeben oder verkaufen, solange diese Programme einen Copyright-Vermerk enthalten. Unter Copyright-Vermerk verstehen wir entweder Ihren eigenen Copyright-Vermerk oder gegebenenfalls einen Hinweis wie »Created using ObjectWindows by Borland International 1991«.

Die Beispielprogramme von ObjectWindows, die auf den Originaldisketten mitgeliefert werden, demonstrieren die verschiedenen Eigenschaften von Obejct-Windows. Sie dienen ausschließlich zu Lehrzwecken. Sie dürfen diese Beispielprogramme editieren oder für den eigenen Gebrauch modifizieren. Sie dürfen diese Programme jedoch nicht als Ganzes oder in Teilen in compilierter Form oder als Quelltext weitergeben oder verkaufen. Sie dürfen die Routinen aus den Beispielprogrammen in eigenen Anwendungsprogrammen verwenden, solange die resultierenden Programme nicht einem Bespielprogramm oder einem Teil eines Beispielprogrammes in Erscheinung oder Funktion gleichen.

Einschränkung der Gewährleistung
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber
Borland GmbH
Gautinger Straße 10
Postfach
8130 Starnberg
Telefon 0 81 51 / 2 63 - 0
Telefax 0 81 51 / 149 09

Umschlag und typografische Konzeption: Rudolf Paulus Gorbach

Übertragung ins Deutsche: Peter Baier, Roland Konstanty, Waltraud Kopelent

Typographische Konzeption des Innenteils: Rudolf Paulus Gorbach

Grafiken: Sid Gastl

Satz: Hanna Bergbauer

PostScript-Belichtung: CD Computer & Dokumentations GmbH, Aalen

Druck und Bindung: Kösel GmbH, Kempten

2., unveränderte Auflage, Mai 1992

Inhalt

Einführung 1

Neue Eigenschaften und Änderungen von Version 3.0 1
Voraussetzungen an Hardware und Software 2
Eine Bemerkung zur Terminologie 3
Übersicht über dieses Handbuch 3
Copyright-Erklärung 4
Die Warenzeichen 5
Haben Sie Fragen oder Probleme? Die technische Beratung von Borland 5
Empfehlenswerte Bücher 6
Bücher zu TDW 6
Bücher über MS-Windows 6

1. Disketten und Installation 9

Die TDW-Disketten 9
Die Datei README 10
Die Datei ASMDEBUG.TDW 10
Die Datei UTILS.TDW 10
Die Installation von TDW 11
Die Installation von TDDEBUG.386 11
Hardware-Debuggen 12
Wie geht's weiter? 12
Neulinge im Umgang mit Turbo C++ 12
Fortgeschrittene im Umgang mit Turbo C++ ohne Erfahrung mit Turbo Debugger für Windows 12
Fortgeschrittene im Umgang mit Turbo Debugger 13

2. Fehlersuche mit Turbo Debugger 15

Liegt überhaupt ein Fehler vor? 15
Wo ist der Fehler? 15
Um was für eine Art von Fehler handelt es sich? 16
Wie kann man den Fehler beheben? 16
Wozu Sie Turbo Debugger einsetzen können 16
Was Turbo Debugger nicht kann 18
Wie Turbo Debugger vorgeht 18

Die Vorzüge von Turbo Debugger 18
Menüs und Dialogfenster 19
Gewußt, wo 21
Geschichtsstunden 24
Sukzessive Auswahl 25
Makros erstellen 25
Schaufensterbummel 26
Einfügen und Kopieren 35
Das Hilfesystem 39

3. Einführung in das Debuggen 43

Das Beispielprogramm 43
TDW im Einsatz 45
Die Menüleiste 45
Die Statuszeile 45
Die Fenster 46
Das C-Beispielprogramm 47
Breakpoints setzen 48
Das Watches-Fenster 49
Einfache Datenobjekte in C untersuchen 50
Zusammengesetzte Datenstrukturen in C untersuchen 51
C-Daten ändern 52

4. Der Aufruf von Turbo Debugger 55

Ein Programm zum Debuggen vorbereiten 55
Turbo C++ Programme vorbereiten 55
Den Debugger starten 56
Kommandozeilenoptionen angeben 56
Turbo Debugger und Windows 58
Kommandozeilenoptionen 58
Konfigurationsdatei laden (-c) 58
Bildschirmumschaltung (-d) 59
Hilfestellungen (-h und -?) 59
Assembler-Modus (-l) 59
Mausunterstützung (-p) 60
Quellcode-Optionen (-s) 60
Startverzeichnis (-t) 60
Konfigurations-Dateien 61
Das Menü Options 61
Das Auswahlfeld Language 61
Das Menü Macros 62
Das Dialogfenster Display options 62
Der Befehl Path for source 64
Der Befehl Save options 64
Der Befehl Restore options 65
Rückkehr zu Windows 65

5. Steuerung der Programmausführung 67

Den aktuellen Zustand des Programms untersuchen 68
Das Fenster Variables 68
Das Fenster Stack 71
Der Befehl Origin im lokalen Menü 72
Der Befehl Get Info 72
Das Menü Run 75
Run 75
Go to cursor 76
Trace into 76
Step over 76
Execute to... 77
Until return 77
Animate... 77
Back trace 78
Instruction trace 78
Arguments... 78
Program reset 78
Das Fenster Execution History 79
Der Ausschnitt mit den ausgeführten Programmschritten 80
Programmausführung unterbrechen 81
Programm beenden 82
Eine Debuggersitzung erneut starten 82
Programm erneut laden 83
Ein neues Programm zum Debuggen laden 83
Die Argumente eines Programms verändern 84

6. Programmdaten untersuchen und ändern 85

Das Menü Data 86
Inspect... 86
Evaluate/ Modify... 86
Add Watch... 89
Function Return 89
Auf Datenobjekte in Quelldateien zeigen 89
Das Fenster Watches 90
Das lokale Menü des Watches-Fensters 91
Inspector-Fenster 92
Inspector-Fenster für C-Daten 92
Inspector-Fenster für Assemblerdaten 95
Das lokale Menü des Inspector-Fensters 98
Range... 98
Change... 98
Inspect 99
Descend 99
New expression... 99
Type Cast... 100

7. Breakpoints 101

Das Menü Breakpoints 102
Toggle 102
At... 103
Changed memory global 103
Expression true global... 103
Hardware breakpoint... 103
Delete all 103
Das Fenster Breakpoints 104
Das lokale Menü des Breakpoint-Fensters 104
Das Dialogfenster Breakpoint Options 107
Das Dialogfenster Conditions and Actions 109
Optionen für Breakpoints 113
Einfache Breakpoints 113
Globale Breakpoints 114
Unterbrechung bei Datenveränderungen 114
Bedingte Breakpoints 115
Hardware-Breakpoints 116
Variablenwerte protokollieren 116
Breakpoints und Templates 117
Das Fenster Log 118
Das lokale Menü des Log-Fensters 119

8. Dateien untersuchen und ändern 121

Quelldateien eines Programms untersuchen 121
Das Fenster Module 122
Das lokale Menü des Module-Fensters 123
Andere Dateien untersuchen 125
Das Fenster File 125
Das lokale Menü des File-Fensters 126

9. Ausdrücke 129

Die Auswahl der Sprache für die Auswertung der Ausdrücke 129
Adressen im Code, Adressen von Daten, Zeilennummern 130
Der Zugriff auf Symbole außerhalb des Gültigkeitsbereichs 131
Die Vorgabe eines Gültigkeitsbereichs 131
Der implizite Gültigkeitsbereich für die Auswertung von Ausdrücken 134
Byte-Folgen 135
C-Ausdrücke 135
C-Symbole 136
Pseudovariablen für die C-Register 136
C-Konstanten und Zahlenformate 137
Escape-Sequenzen in C 137
C-Operatoren und deren Rangfolge 138
Die Ausführung von C-Funktionen in Ihrem Programm 139

C-Ausdrücke mit Seiteneffekten 139
Reservierte Wörter in C und die explizite Datentypvorgabe 140
Assembler-Ausdrücke 141
Symbole in Assembler 141
Assembler-Konstanten 141
Operatoren in Assembler 142
Die Formatsteuerung 142

10. Objektorientiertes Debuggen 145

Das Fenster Hierarchy 145
Die Listenausschnitt mit den Objekttypen 146
Der Ausschnitt mit dem Objektbaum 147
Der Ausschnitt mit dem Vorfahrenbaum 147
Das Inspector-Fenster für Objekttypen/Klassen 148
Die lokalen Menüs des Inspector-Fensters für Objekttypen/Klassen 149
Inspector-Fenster für Objekt-/Klasseninstanzen 150
Die lokalen Menüs des Inspector-Fensters für Objekt- und Klasseninstanzen 150
Der mittlere und der untere Ausschnitt 152

11. Debuggen unter Windows 153

Spezialitäten für Windows 153
Protokollierung von Windows-Botschaften 154
Speicher- und Modullisten 161
Das Debuggen von dynamischen Link-Bibliotheken (DLLs) 164
Die Umwandlung von Speicherhandles in Adressen 169

12. Debuggen auf Assembler-Ebene 171

Grenzen des Debuggens auf Quellcode-Ebene 171
Das CPU-Fenster 172
Der Code-Ausschnitt 173
Der Disassembler 174
Das lokale Menü des Code-Ausschnitts 175
Die Register- und Statusausschnitte 178
Das lokale Menü des Registerausschnitts 178
Das lokale Menü des Statusausschnitts 179
Der Selector-Ausschnitt 180
Das lokale Menü des Selector-Ausschnitts 180
Der Datenausschnitt 181
Das lokale Menü des Datenausschnitts 182
Der Stack-Ausschnitt 185
Das lokale Menü des Stack-Ausschnitts 185
Der Assembler 186
Vorgaben für die Größe von Adreßoperanden 186
Vorgaben für den Datentyp eines Operanden 187
Stringbefehle 187

Das Fenster Dump 188
Das Fenster Registers 188

13. 80x87-Koprozessor und Gleitkomma-Emulator 191

Der 80x87-Koprozessor im Vergleich zum Emulator 191
Das Fenster Numeric Processor 191
Der Registerausschnitt 192
Der Controlausschnitt 193
Der Statusausschnitt 194

14. Die Befehlsübersicht 195

Funktionstasten und Tastenkürzel 195
Die Befehle der Menüleiste 197
Das System-Menü 198
Das File-Menü 198
Das Edit-Menü 198
Das View-Menü 198
Das Run-Menü 199
Das Breakpoints-Menü 199
Das Data-Menü 199
Das Options-Menü 200
Das Window-Menü 200
Das Help-Menü 200
Die Befehle der lokalen Menüs 201
Das lokale Menü des Breakpoints-Fensters 201
Die Menüs des CPU-Fensters 201
Das Fenster Dump 203
Die lokalen Menüs des Fensters Execution History 204
Das File-Fenster 204
Das lokale Menü des Log-Fensters 204
Das Module-Fenster 205
Die lokalen Menüs des Fensters Windows Messages 205
Das Clipboard-Fenster 206
Die lokalen Menüs des Fensters Numeric processor 206
Das Hierarchy-Fenster 207
Das Registers-Fenster 207
Das lokale Menü des Stack-Fensters 207
Das Variables-Fensters 208
Das Fenster Watches 208
Das lokale Menü des Inspector-Fensters 209
Das Inspector-Fenster für Objekttypen/Klassen 209
Das Inspector-Fenster für Objekt-/Klasseninstanzen 209
Die Textausschnitte 210
Die Listenausschnitte 211
Funktionstasten in Eingabefeldern und Eingabeaufzeichchnungsfeldern 211

Befehle zum Verändern der Fensteranordnung 212
Die Jokerzeichen 213
Der vollständige Menübaum 213

15. Ein Standard C-Programm debuggen 217

Wenn's nicht läuft 217
Der persönliche Stil beim Debuggen 218
Einfach mal laufen lassen 218
Schrittweise testen 218
Mögliche Fehler 219
Allgemeine Fehler 219
Typische Fehler in C 220
Richtigkeit überprüfen 225
Grenzfälle und Einschränkungen 225
Falsche Eingabedaten 225
Leere Eingabe 226
Die Fehlersuche als Teil der Programmentwicklung 226
Beispiel für eine Debugger-Sitzung 226
Fehler suchen 227
Strategie festlegen 227
Der Aufruf von TDW 228
Untersuchungen 228
Breakpoints 229
Das Watches-Fenster 229
Das Dialogfenster Evaluate/Modify 230
Es ist vollbracht! 230

16. Ein ObjectWindows-Programm debuggen 233

Über das Programm 233
Die Definition des Fenstertyps ScribbleWindow 234
Das Programm erstellen 236
Das Programm debuggen 236
Den ersten Fehler suchen 237
Den Farbzuordnungsfehler lokalisieren 239
Lokalisieren des Fensterbereichsfehlers 243
Den Fehler beim Bildschirmlöschen lokalisieren 246

A. Kommandozeilenoptionen 249

B. Inline-Assembler-Befehle 251

C. Die Konfiguration von Turbo Debugger 255

Der Aufruf von TDWINST 256
Einstellung der Bildschirmfarben 256

Die Änderung der Bildschirmfarben 256
Die Standardfarben 258
Die Einstellung der Bildschirmparameter in TDW 258
Display swapping 259
Integer format 259
Beginning Display 260
Screen lines 260
Tab size 260
Maximum tiled watch 260
Fast screen update 260
Permit 43/50 line mode 261
International sort 261
User screen updating 261
Log list length 262
Floating precision 262
Range Inspect 262
Die Einstellung der TDW-Optionen 262
Directories 262
Input & Prompting 263
Source Debugging 264
Das Einstellen des Bildschirmmodus 265
Kommandozeilenoptionen und Installationsoptionen 266
Wenn alles wie gewünscht eingestellt wurde 267
Speichern der Änderungen 267
TDWINST verlassen 268

D. Dialogfenster und Fehlermeldungen 269

Meldungen in Dialogfenstern 269
Die Fehlermeldungen 277
Schwerwiegende Fehler 277
Andere Fehlermeldungen 278

E. Glossar 293

Index 297

Einführung

Turbo Debugger für Windows (TDW) ist ein Quelltext-Debugger, der für Programmierer entwickelt wurde, die mit Turbo-C++ für Windows arbeiten.

Einander überlappende Fenster, Pull-Down- und Pop-Up-Menüs und umfassende Mausunterstützung sind einige der Kennzeichen der komfortablen Benutzeroberfläche. Ein kontextbezogenes Hilfesystem begleitet Sie bei allen Operationen. Einige der Eigenschaften:

- Debuggen von MS-Windows-Anwendungen
- Auswertung aller C-, C++ und Assembler-Ausdrücke
- frei konfigurierbarer Bildschirmaufbau
- voller Zugriff auf Assembler und CPU
- mächtige Breakpoint-Befehle mit Protokollfunktion
- Rückwärts-Ausführung
- Unterstützung der objektorientierten Programmierung in Turbo C++ für Windows
- arbeitet im Textmodus

Neue Eigenschaften und Änderungen von Version 3.0

Die Version 3.0 von Turbo Debugger weist einige Erweiterungen auf, die sich auf das Debuggen von Windows-Programmen beziehen. Neue Eigenschaften, die unter Windows zur Verfügung stehen sind:

- Mit Hilfe der Zwischenablage können Sie Texte aus Windows-Anwendungen kopieren und in Dialogfenster Eingabefelder und in andere Fenster einfügen. Weitere Informationen erhalten Sie in Kapitel 2, »Fehlersuche mit Turbo Debugger«.

■ Es gibt einige neue Optionen für Breakpoints (Kapitel 7 »Breakpoints«), mit denen Sie:
∩ mehrere Bedingungen und Aktionen für einen Breakpoint vorgeben können;
∩ Gruppen von Breakpoints setzen und entfernen können;
∩ Breakpoints für alle Funktionen oder Prozeduren eines Moduls setzen oder entfernen können;
∩ Breakpoints bei allen Methoden eines Objekttyps oder Elementfunktionen einer Klasse setzen oder entfernen können.

■ Es werden C++-Templates und verschachtelte Klassen unterstützt.

■ Die internationale Sortierordnung kann mit Hilfe der Spracheinstellung unter Windows ausgewählt werden. Sie sollten diese Möglichkeit mit Hilfe des Konfigurationsprogramms TDWINST.EXE (siehe TDWINST.DOC) einschalten.

■ Das CPU-Fenster hat einen neuen Ausschnitt, das die Protected-Mode-Selektoren und den Inhalt der Speicherbereiche anzeigt, die durch diese Selektoren adressiert werden.

■ Der Gerätetreiber TDDEBUG.386 unterstützt die Hardware-Debug-Register des Intel-80386-Prozessors (und nachfolgender Prozessoren). In Kapitel 2, »Disketten und Installation«, finden Sie nähere Informationen über die Installation von TDDEBUG.386. Informationen über Hardware-Debuggen finden Sie inder Datei HDWDEBUG.TDW.

■ Das Debuggen von DLLs wurde erheblich beschleunigt. TDW lädt nun in einem Schritt sowohl die Sympboltabelle eines Anwendungsprogramms als auch die Symboltabelle jeder DLL, die Sie explizit laden bzw. auf deren Code Sie vom Anwendungsprogramm aus zugreifen (siehe Kapitel 9, »Ausdrücke«).

Voraussetzungen an Hardware und Software

TDW stellt dieselben Anforderungen an Hardware und Software wie Turbo C++ für Windows auch.

TDW unterstützt den Betrieb mit Super-VGAs über eine spezielle DLL, TDVIDEO.DLL. Für die verschiedenen Grafikadapter werden eine Reihe von DLLs mitgeliefert (siehe Datei README). Um diese DLLs für TDW zu verwenden, kopieren Sie die entsprechende Datei in dasselbe Verzeichnis wie TDW.EXE und geben ihr den Namen TDVIDEO.DLL.

Für den Einsatz von TDW benötigen Sie Turbo C++ für Windows. Zudem müssen Sie Ihren Quellcode zusammen mit den Debug-Informationen in eine ausführbare Datei (».EXE«-Datei) compiliert haben.

Wenn Sie TDW aufrufen, benötigen sowohl die ».EXE«-Datei als auch die ursprünglichen Quellcodedateien. TDW sucht die Quellcodedateien zuerst in dem Verzeichnis, in dem diese vom Compiler beim Compilieren gefunden wurden, anschließend in den Verzeichnissen, die mit dem Befehl *Options/Path*

for Source angegeben wurden, daraufhin im aktuellen Verzeichnis und zuletzt in dem Verzeichnis, in dem sich die »-EXE«-Datei befindet.

Eine Bemerkung zur Terminologie

In diesem Handbuch werden manche Begriffe mit einer etwas allgemeineren Bedeutung verwendet. Dazu zählen Begriffe wie Modul und Argument.

Modul Ein Modul wird in diesem Handbuch im Sinne eines C- oder Assembler-Moduls verwendet, umfaßt aber auch die Unit in Pascal.

Argument Der Begriff *Argument* ist, so wie er in diesem Handbuch verwendet wird, austauschbar mit dem Begriff Parameter. Er bezieht sich sowohl auf Kommandozeilenargumente als auch auf Argumente von Funktionen und Prozeduren.

Übersicht über dieses Handbuch

Hier finden Sie eine kurze Zusammenfassung der Kapitel und Anhänge in diesem Handbuch:

Kapitel 1 »Disketten und Installation« enthält eine Aufstellung aller ausgelieferten Dateien auf den Disketten und eine Beschreibung der Installation von TDW, sowie einen kurzen, nach Ihren Interessen aufgegliederten Wegweiser durch dieses Handbuch.

Kapitel 2 »Fehlersuche mit Turbo Debugger« erläutert die Benutzeroberfläche, die Menüs und die Fenstertechnik von TDW. Des weiteren wird Ihnen gezeigt, wie Eingabeaufforderungen und Fehlermeldungen zu beantworten sind.

Kapitel 3 »Einführung in das Debuggen« erklärt zunächst einige wichtige Befehle, die daraufhin anhand eines Beispielprogramms in Turbo C++ ausführlich erläutert werden.

Kapitel 4 »Der Aufruf von Turbo Debugger« enthält eine detaillierte Beschreibung der Kommandozeilenoptionen sowie der Konfigurationsdateien.

Kapitel 5 »Steuerung der Programmausführung« ist eine Beschreibung der *Run*-Befehle.

Kapitel 6 »Programmdaten untersuchen und ändern« beschreibt das einzigartige Konzept des *Inspector*-Fensters.

Kapitel 7 »Breakpoints« enthält eine umfassende Beschreibung der Einsatzmög-

lichkeiten von Breakpoints in Turbo Debugger. Es wird gezeigt wie Sie Bedingungen und Aktionen für Breakpoints festlegen können.

Kapitel 8 »Dateien untersuchen und ändern« beschreibt, wie man Programm- und sonstige Dateien untersuchen und ändern kann.

Kapitel 9 »Ausdrücke« enthält eine vollständige Beschreibung der Syntax von C- und Assembler-Ausdrücken sowie die Formatierung der ausgegebenen Daten.

Kapitel 10 »Objektorientiertes Debuggen« erklärt die speziellen Funktionen des Debuggers für die Untersuchung von Objekten eines Programms von Turbo C++ für Windows.

Kapitel 11 »Debuggen unter Windows« beschreibt die speziellen Möglichkeiten von TDW zum Debuggen unter Windows.

Kapitel 12 »Debuggen auf Assembler-Ebene« erläutert die Arbeit im CPU-Fenster. Weitere Informationen finden Sie in der Datei ASMDEBUG.TDW.

Kapitel 13 »80x87 und Fließkomma-Emulator« behandelt die Möglichkeiten des Numeric-Fensters, um Fließkomma-Operationen zu debuggen.

Kapitel 14 »Die Befehlsübersicht« enthält eine tabellarische Zusammenfassung der Befehle in Turbo Debugger.

Kapitel 15 »Ein Standard-C-Programm debuggen« stellt zunächst die wichtigsten Fehlerquellen in der Programmierung vor und zeigt dann anhand eines Praxisbeispiels, wie man diesen Fehlern auf die Schliche kommen kann..

Kapitel 16 »Ein ObjcetWindows-Programm debuggen« zeigt, ebenfalls anhand eines Beispielprogramms, wie Fehler in einem Programm zu finden sind, das die Objekte der *ObjectWindows* -Bibliothek einbindet.

Anhang A »Kommandozeilenoptionen« enthält eine Aufstellung aller Optionen, die beim Aufruf von Turbo Debugger verwendet werden können.

Anhang B »Inline-Assemblerbefehle« faßt die Befehle und reservierten Wörter des integrierten Assemblers zusammen.

Anhang C »Die Konfiguration von Turbo Debugger« beschreibt alle Optionen, die mit dem Installationsprogramm TDWINST eigestellt werden können.

Anhang D »Dialogfenster und Fehlermeldungen« listet alle Ausgaben von Fehlermeldungen bis hin zu Eingabeaufforderungen vollständig auf.

Anhang E »Glossar« erklärt die wichtigsten Fachausdrücke.

Copyright-Erklärung

Jede Vervielfältigung dieses Handbuchs sowie des Turbo-Debugger-Softwareprogrammes wird strafrechtlich verfolgt. Die Rechte an dem Buch und

die Rechte an dem Softwareprogramm Turbo Debugger für Windows liegen bei Borland International.

Der rechtmäßige Erwerb der Programmdisketten und der Dokumentation erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Turbo Debugger für Windows nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.

Die Warenzeichen

Wir nehmen in diesem Buch auf geschützte Warenzeichen Bezug, die innerhalb des laufenden Textes nicht mehr explizit als solche gekennzeichnet sind – aus dem Fehlen einer Kennzeichnung kann also nicht geschlossen werden, daß der entsprechende Produktname frei von Rechten Dritter ist:

- IBM, AT, PS/2 sind geschützte Warenzeichen von International Business Machines.
- Microsoft und MS-DOS sind geschützte Warenzeichen von Microsoft.

Haben Sie Fragen oder Probleme? Die technische Beratung von Borland

Bitte lesen Sie die Handbücher und die Datei README sorgfältig durch. Wenn danach Fragen offen geblieben sind und Sie technische Beratung brauchen, sollten Sie die folgenden Informationen parat haben:

1. das deutsche Handbuch;

2. den Produktnamen und die Versionsnummer; (Die Versionsnummer erscheint, wenn Sie TDW ohne Optionen aufrufen oder *About* im Menü *System* auswählen.)

3. Produktnamen, Modellbezeichnung Ihres Computers und gegebenenfalls der Zusatz-Hardware;

4. Betriebssystem und dessen Versionsnummer;

5. den Inhalt der Dateien AUTOEXEC.BAT, CONFIG.SYS, WIN.INI und SYSTEM.INI.

Technische Beratung erhalten Sie:

1. bei Ihrem Softwarehändler oder einem Benutzerclub;

2. bei schriftlichen Anfragen durch unsere technische Beratung;

3. in dringenden Fällen telefonisch über die Borland Hotline;
   Montag - Donnerstag 14.00 - 17.00 Uhr, Freitag 14.00 - 16.00 Uhr
   Telefon (08151) 263-303 für TDW für C++
   -302 für TDW für Pascal

Empfehlenswerte Bücher

Viele Verlage unterstützen die Borland-Produkte durch eine Reihe exzellenter Bücher, die sowohl für Programmierneulinge als auch fortgeschrittene Anwender von Nutzen sind. Darüber hinaus sollten Sie noch Bücher über andere Themen, wie z.B. die Programmierung für Windows, lesen.

Bücher zu TDW

Nachstehend finden Sie eine Liste von Titeln, die zusätzliche Informationen über Turbo Debugger und seine Tools bieten:

Ackermann, Charles. *Turbo Debugger and Tools: A Self-Teaching Guide* , John Wiley and Sons (New York: 1990).

Monjademi, Peter. *Turbo Assembler* , Addison Weseley, ISBN 3-89319-255-5

Murray and Pappas. *Asseblierungsprogrammierung f201r Experten*, Mc Graw-Hill Book Company, US-ISBN: 0-07-881217-8, deutsch-ISBN 3-89028-108-7

Swan, Tom. *Mastering Turbo Debugger and Tools,* Howard W. Sams and Co. (Carmel, IN: 1990).

Syck, Gary. *The Waite Group's Turbo Assembler Bible,* Howard W. Sams and Co. (Carmel, IN: 1990).

Bücher über MS-Windows

Die folgenden Bücher bieten Informationen über das Schreiben von Windows-Anwendungsprogrammen:

Microsoft staff. *Microsoft Windows Software Development Kit: Guide to Program - ming,* Microsoft Corporation (Redmond, WA: 1990).

Microsoft staff. *Microsoft Windows Software Development Kit: Reference, Vols. 1 and 2,* Microsoft Corporation (Redmond, WA: 1990).

Microsoft staff. *Microsoft Windows Software Development Kit: Tools,* Microsoft Corporation (Redmond, WA: 1990).

Petzold, Charles. *Programming Windows,* Microsoft Press (Redmond, WA: 1990).

1. Disketten und Installation

Turbo Debugger für Windows (TDW) ist ein Bestandteil von Turbo C++ für Windows. Die ausgelieferten Disketten enthalten alle Programme, Dateien und Hilfsprogramme, die Sie für die Fehlersuche in Programmen benötigen, die mit Turbo C++ für Windows geschrieben wurden.

Das *Benutzerhandbuch* zu *Turbo Debugger f201r Windows* enthält eine nach Themen eingeteilte Einführung in die Möglichkeiten von TDW sowie eine vollständige Befehlsübersicht.

Bitte machen Sie, bevor Sie beginnen, mit Turbo Debugger zu arbeiten, eine Arbeitskopie der ausgelieferten Disketten und verwahren Sie diese Originaldisketten an einem sicheren Ort. Benützen Sie die Originaldisketten nur als Sicherung und erledigen Sie Ihre Arbeit mit TDW mit Hilfe der Kopie. Die Originaldisketten sind Ihre einzige Sicherung, falls Ihre Dateien verloren gehen sollten.

Falls Sie mit der Lizenzvereinbarung von Borland noch nicht vertraut sind, sollten Sie jetzt die entsprechende Stelle in der Einleitung lesen und Ihre Registrationskarte ausfüllen und abschicken. (Sie werden dann immer automatisch informiert, wenn neue Versionen und Produkte erscheinen.)

Die TDW-Disketten

Bei der Installtion von Turbo C++ für Windows werden alle Dateien auf Ihre Festplatte kopiert. Sie rufen lediglich das einfach zu bedienende Installationsprogramm INSTALL.EXE von der Installationsdiskette auf. Dieses Program kopiert nicht nur die Dateien von Turbo C++ für Windows, sondern auch alle Dateien zu TDW.

Eine Liste der ausgelieferten Dateien finden Sie in der Datei FILELIST.DOC auf der Installationsdiskette.

Die Datei README

Es ist sehr wichtig, daß Sie sich die Zeit nehmen, die Datei README auf der Installationsdiskette durchzulesen, bevor Sie irgendetwas anderes mit TDW machen. Diese Datei enthält Informationen, die in allerletzter Minute zusammengestellt wurden und in diesen Handbuch nicht enthalten sind. Außerdem sind alle Dateien beschrieben, die auf den Originaldisketten ausgeliefert wurden. (Mit dem Editor von Turbo C++ für Windows oder dem Windows-Programm *Notizblock* können Sie sich den Inhalt der Datei README anzeigen lassen.

Die Datei ASMDEBUG.TDW

Diese Datei enthält Informationen über das Debuggen von Programmen, die in Turbo Assembler geschrieben wurden. Sie finden hier auch sehr viele nützliche Hinweise zum Debuggen von Inline-Code.

Die Datei UTILS.TDW

Diese Datei beschreibt die Kommandozeilenprogramme, die zusammen mit TDW ausgeliefert werden. Diese Hilfesprogramme sind, TDSTRIP und TDUMP.

Eine kurze Beschreibung der einzelnen Hilfsprogramme für TDW:

- TDSTRIP.EXE dient zum Entfernen der Debug-Informationen (der Symboltabelle) aus Programmen, ohne daß das Programm erneut gelinkt werden muß. Sie können diese Operation mit einer ».COM«-Datei durchführen und die Symboltabelle in einer ».TDS«-Datei speichern, um die ».COM«-Datei debuggen zu können.

  Dazu compilieren Sie den Quellcode mit der Option *Debugging Information* als ».EXE«-Datei mit nur einem Segment, und rufen Sie anschließend TDSTRIP mit der ».EXE«-Datei als Argument auf. Wenn die ».EXE«-Datei in eine ».COM«-Datei umgewandelt werden kann, erzeugt TDSTRIP eine ».TDS«-Datei und eine ».COM«-Datei. Mit Hilfe der ».TDS«-Datei kann die ».COM«-Datei im Quellcode untersucht werden.

- TDUMP.EXE zeigt den Inhalt von Objektmodulen und ».EXE«-Dateien in einem lesbaren Format an.

Hinweis: Um eine Liste der verfügbaren Kommandozeilenoptionen zu den Programmen TDSTRIP.EXE oder TDUMP.EXE zu erhalten, geben Sie nur den

Programmnamen ein und drücken ↵ Für die Kommandozeilenoptionen von TDUMP.EXE würden Sie zum Beispiel folgenden Befehl eingeben:

```
TDMAP
```

Die Installation von TDW

Mit dem Programm INSTALL.EXE von Turbo C++ für Windows wird auch TDW installiert. Während der Installation erzeugt INSTALL.EXE eine Windows-Programmgruppe und legt darin die Programmsymbole für Turbo C++ für Windows sowie Turbo Debugger ab. Allgemeine Hinweise zur Installation finden Sie in der Datei README.

Die Installation von TDDEBUG.386

Auf den Installationsdisketten befindet sich die Datei TDDEBUG.386, die dieselben Funktionen erfüllt wie die Datei WINDEBUG.386 des Windows-Entwicklungspakets. Damit werden Hardware-Debugger-Register des Intel Prozessors 80386 (und nachfolgender Prozessoren) unterstützt.

Das Installationsprogramm sollte diese Datei auf Ihre Festplatte kopieren und die Windows-Datei SYSTEM.INI so verändern, daß TDDEBUG.386 anstelle von WINDEBUG.386 geladen wird. Wenn das Installationsprogramm diese Aufgabe nicht ausführen kann, erhalten Sie eine entsprechende Meldung. Sie müssen diesen Vorgang dann Schritt für Schritt durchführen:

1. Das Installationsprogramm hat TDDEBUG.386 von der Installationsdiskette auf Ihre Festplatte kopiert. Das Standardverzeichnis für diese Datei ist C:\TCWIN\BIN. Wenn Sie diese Datei in ein anderes Verzeichnis verlagern wollen, ersetzen Sie das Verzeichnis in den Anweisungen.

2. Mit Hilfe eines Editors öffnen Sie die Windows-Datei SYSTEM.INI, suchen den Abschnitt [386enh] und fügen folgende Zeile in diesen Abschnitt ein:

```
device=c:\tcwin\bin\tddebug.386
```

3. Wenn sich im Abschnitt [386enh] ein Eintrag befindet, mit dem WINDEBUG.386 geladen wird, wandeln Sie diesen Eintrag mit einem vorangestellten Strichpunkt in einen Kommentar um oder löschen die ganze Zeile (TDDEBUG.386 und WINDEBUG.386 dürfen nicht gemeinsam geladen werden).

Wird WINDEBUG.386 zum Beispiel aus dem Verzeichnis C:\WINDOWS geladen, muß die in einen Kommmentar umgewandelte Zeile wie folgt lauten:

```
;device=c:\windows\windebug.386
```

Hardware-Debuggen

Zum Debuggen von Windows-Programmen können Sie die Hardware-Debugger-Register des Intel-80386-Prozessors (oder nachfolgender Prozessoren) verwenden. Dazu müssen Sie TDDEBUG.386 laden, wenn Sie Windows aufrufen.

In der Datei HDWDEBUG.TD finden Sie weitere Informationen zum Debuggen von Windows-Programmen mit Hilfe der Hardware-Debugger-Register.

Wie geht's weiter?

Sie können nun beginnen, sich mit TDW vertraut zu machen. Da dieses Handbuch für zwei verschiedene Benutzergruppen geschrieben ist, werden ganz verschiedene Kapitel für Sie von besonderem Interesse sein. Die folgende Übersicht wird Ihnen bei der Orientierung helfen.

Neulinge im Umgang mit Turbo C++

Wenn Sie Anfänger sind, werden Sie erst einmal lernen wollen, kleine Programme mit Turbo C++ für Windows zu schreiben, bevor Sie den Umgang mit dem Debugger erlernen. Es gibt natürlich keinen besseren Weg, sich Wissen über den Debugger anzueignen, als einen tatsächlichen Fehler aufzuspüren, mit dem Sie in Ihrer eigenen Arbeit konfrontiert sind. Am besten gehen Sie wie folgt vor: Nachdem Sie sich ein Grundwissen in Ihrer Sprache angeeignet haben, arbeiten Sie Kapitel 3, »Einführung in das Debuggen«, durch, eine Einführung in die wichtigsten Funktionen von TDW. Sie lernen dort genügend über die Möglichkeiten zum Debuggen Ihres ersten Programms. Die weiterführenden Möglichkeiten des Debuggers werden dann in einem späteren Kapitel behandelt.

Fortgeschrittene im Umgang mit Turbo C++ ohne Erfahrung mit Turbo Debugger für Windows

Wenn Sie bereits Erfahrung im Programmieren mit Turbo C++ haben, aber mit TDW noch nicht vertraut sind, können Sie sich in Kapitel 2, »Fehlersuche in TDW«, über die interessanten, neuen Möglichkeiten der Turbo-Debugger-Benutzerschnittstelle informieren. Sie können dann, je nach persönlichem Arbeitsstil, entweder das Tutorial in Kapitel 3, »Einführung in das Debuggen«, durcharbeiten, oder gleich zu Kapitel 4, »Der Aufruf von Turbo Debugger«,

übergehen. Eine Gesamtübersicht zu allen Befehlen finden Sie in Kapitel 14, »Die Befehlsübersicht«.

Fortgeschrittene im Umgang mit Turbo Debugger

Wenn Sie Turbo Debugger bereits in der Vergangenheit verwendet haben, sind Sie wahrscheinlich mit den wichtigsten Eigenschaften von Turbo Debugger gut vertraut. In diesem Fall sollten Sie sich Kapitel 11, »Debuggen unter Windows«, zuwenden, in dem alle speziellen Möglichkeiten von TDW zur Fehlersuche in Windows-Programmen erläutert werden. Ein weiteres Kapitel, das Sie als sehr hilfreich empfinden werden, ist Kapitel 15, »Debuggen einer ObjectWindows-Applikation«, in dem Sie durch eine Debugger-Sitzung mit einer Windows-Applikation geführt werden, in der die OnjectWindows-Bilbliothek verwendet wird.

2. Fehlersuche mit Turbo Debugger

Debuggen ist das Auffinden und Korrigieren von Fehlern (»bugs«) in Programmen. Es ist nichts außergewöhnliches, mehr Zeit auf die Fehlersuche in einem Programm zu verwenden, als für die gesamte übrige Programmerstellung. Debuggen ist keine exakte Wissenschaft. Das beste Mittel zum Debuggen ist häufig immer noch der eigene Grips. Nichtsdestoweniger kann man sich die Aufgabe durch eine systematische Vorgehensweise erleichtern.

Der Prozeß der Fehlersuche gliedert sich im großen und ganzen in vier Schritte.

1. Einen Fehler erkennen.
2. Den Fehler lokalisieren.
3. Die Fehlerursache finden.
4. Den Fehler korrigieren.

Liegt überhaupt ein Fehler vor?

Der erste Punkt kann sehr offensichtlich sein. Der Computer hängt sich auf, sobald Sie das Programm laufen lassen. Oder er stürzt in einem Schwall von sinnlosen Zeichen ab. Manchmal jedoch sind die Probleme nicht so leicht zu erkennen. Das Programm kann wunderbar laufen, bis eine bestimmte Zahl eingegeben wird, etwa 0 oder eine negative Zahl, oder bis Sie sich das Ergebnis genau ansehen. Möglicherweise bemerken Sie erst dann, daß das Ergebnis vielleicht um einen Faktor von 0,2 nicht stimmt, oder daß die Initialen in einer Namensliste falsch sind.

Wo ist der Fehler?

Der zweite Schritt ist manchmal der schwierigste: den Fehler zu lokalisieren. Sehen wir den Tatsachen ins Auge, man kann einfach kein ganzes Programm auf einmal im Kopf behalten (außer es handelt sich um ein außergewöhnlich kleines). Deshalb ist es am Besten, wenn Sie das Programm in Module aufteilen

und diese einzeln debuggen. Die strukturierte Programmierung ist für diese Art der Fehlersuche ideal.

Um was für eine Art von Fehler handelt es sich?

Der dritte Schritt, das Ermitteln der Fehlerursache, ist wohl der zweitschwierigste von allen. Wenn man erst einmal herausgefunden hat, wo der Fehler ist, ist es im allgemeinen etwas einfacher zu ermitteln, warum das Programm Zicken macht. Wenn Sie zum Beispiel herausgefunden haben, daß der Fehler in einer Prozedur `Ausgabe_der_Namen` liegt, müssen Sie nur noch die Zeilen dieser Prozedur untersuchen und nicht mehr das ganze Programm. Nichtsdestoweniger kann der Fehler noch schwer zu ermitteln sein und die Suche nach der Ursache noch einiges Experimentieren erfordern.

Wie kann man den Fehler beheben?

Der letzte Schritt ist schließlich die Korrektur. Bewaffnet mit Ihren Kenntnissen der Programmiersprache und dem Wissen, wo der Fehler liegt, wetzen Sie diesen aus. Nun starten Sie Ihr Programm nochmal, warten auf den nächsten Fehler und fangen wieder von vorne an.

Häufig wird dieser vierstufige Prozess beim Erstellen eines Programms von selbst durchgeführt. Es gibt zum Beispiel eine Menge Syntaxfehler, die das Compilieren eines Programms verhindern, wenn man sie nicht korrigiert. Turbo C++für Windows verfügt über die Möglichkeit von Syntaxprüfungen, durch die man auf diese Art von Fehlern aufmerksam gemacht wird, damit sie auf der Stelle entfernt werden können. (In Kapitel 15, »Debuggen einer Standard-C-Applikation«, finden Sie eine ausführlichere Behandlung des Ablaufs der Fehlersuche.)

Andere Fehler sind allerdings um einiges heimtückischer und schwieriger zu erkennen. Sie lauern so lange in ihrem Versteck, bis beispielsweise eine negative Zahl eingegeben wird, oder sie sind so schwer zu fassen, daß man einfach ratlos ist. Hier tritt TDW auf den Plan.

Wozu Sie Turbo Debugger einsetzen können

Mit TDW haben Sie einen Debugger, der wesentlich leistungsfähiger ist als es ein in eine Entwicklungsumgebung eingebauter Debugger je sein könnte.

TDW kann für jedes Programm verwendet werden, das mit Turbo C++ für Windows erstellt wurde. TDW läuft im Textmodus und erlaubt Ihnen, zwischen Ihrem Programm und dem Debugger umzuschalten.

Sie können TDW als Hilfsmittel während der beiden schwierigsten Schritte des Korrekturablaufs verwenden – der Fehlersuche und dem Auffinden der Fehlerursache.

TDW hilft Ihnen mit seinen umfangreichen Möglichkeiten, diese Hürden zu überwinden, die Programmausführung zu verlangsamen und das Programm an jeder beliebigen Stelle zu überprüfen. Man kann sogar neue Werte von Variablen testen, um zu prüfen, wie diese das Programm beeinflussen. Diese Möglichkeiten bedeuten in der Praxis, daß man nach Herzenslust *tracen*, d. h. den Code *Schritt f201r Schritt abarbeitenDaten anschauen* sowie *Variablen untersuchen* und *ändern* kann.

Spur verfolgen (Trace)	Das Programm wird Zeile für Zeile abgearbeitet.
Zurück-verfolgen (Back Tracing)	Sie können bereits ausgeführten Code Schritt für Schritt zurückgehen und so die Anweisungen des Programms zurücknehmen.
Schritte (Step)	Wenn Sie schrittweise einzelne Befehle ausführen, können alle Prozedur- oder Funktionsaufrufe übersprungen werden. Falls Sie sicher sind, daß Ihre Prozeduren und Funktionen fehlerfrei laufen, kann die Fehlersuche auf diese Art beschleunigt werden.
Darstellen (View)	Sie können mit TDW ein spezielles Fenster öffnen, in denen Sie ein Dutzend verschiedener Dinge darstellen können: Variablen, deren Werte, Breakpoints, den Inhalt des Stacks, ein Log-Protokoll, eine Datei mit Daten, eine Quelldatei, den CPU-Code, Speicherbereiche, Register, den Status des Arithmetikprozessors oder Programmausgaben.
Untersuchen (Inspect)	Sie können mit TDW auch die tieferen Zusammenhänge zwischen den Datenstrukturen in Ihrem Programm nachvollziehen, indem Sie den Inhalt von Array-Elementen und Strukturkomponenten mit Hilfe von Inspector-Fenstern ermitteln.
Verändern (Change)	Man kann den Wert einer Variablen sowohl global als auch lokal durch einen bestimmtem Wert ersetzen.
Beobachten (Watch)	Sie können Programmvariablen herausstellen und über ihre sich ändernden Werte auf dem laufenden bleiben, während das Programm weiterläuft.

Mit derart leistungsfähigen Hilfsmitteln kann man ein Programm in kleine Abschnitte aufteilen und dann nachprüfen, ob ein solcher Teilbereich in Ordnung ist, bevor man den nächsten untersucht. Auf diese Weise können Sie sich durch ein Programm arbeiten, unabhängig wie groß oder kompliziert es ist, bis der Fehler gefunden ist. Vielleicht stoßen Sie dabei auf eine Funktion, in der versehentlich einer Variablen ein Wert zugewiesen wird, vielleicht bleibt das Programm auch in einer Endlosschleife hängen, oder es wird in eine chaotische Rekursion gezogen. Wo auch immer das Problem liegt, TDW ist Ihnen eine wesentliche Hilfe beim Auffinden und Identifizieren des Fehlers.

Mit TDW können Sie auch objektorientierte Programme debuggen, die in C++ erstellt wurden. Das Programm kann mit Objekten und Klassen umgehen und ist in der Lage, die dynamische Bindung virtueller Elementfunktionen richtig zu interpretieren, so daß automatisch immer der richtige Code ausgeführt und dargestellt wird.

Was Turbo Debugger nicht kann

Bei all den Möglichkeiten, die TDW bietet, könnte man den Eindruck bekommen, der Debugger könne einfach alles. In Wirklichkeit gibt es mindestens drei Dinge, die TDW nicht für Sie erledigen kann:

- TDW kann Ihr Programm nicht neu compilieren. Dafür müssen Sie den Compiler von Turbo C++ für Windows verwenden.
- TDW kann nicht im Grafikmodus unter Windows betrieben werden, sondern nur im Textmodus.
- TDW erspart Ihnen nicht das Denken. Wenn Sie ein Programm debuggen, ist Nachdenken Ihr wichtigstes Kapital. TDW ist ein leistungsfähiges Werkzeug, gedankenlos angewandt, werden Sie sich jedoch kaum Zeit oder Mühe sparen.

Wie Turbo Debugger vorgeht

Das wirklich Gute ist: TDW bietet höchste Leistungsfähigkeit, während er gleichzeitig leicht, beinahe instinktiv zu bedienen ist. Gekrönt wird diese kunstvolle Mischung aus Leistung und Einfachheit durch eine exzellente Benutzerschnittstelle, über deren Vorteile Sie im nächsten Abschnitt informiert werden.

Die Vorzüge von Turbo Debugger

Haben Sie erst einmal angefangen, mit TDW zu arbeiten, werden Sie nicht mehr davon loskommen. Das Programm wurde speziell für leichtes und bequemes Arbeiten entwickelt. Um dies zu gewährleisten, bietet TDW folgende Möglichkeiten:

- Bequeme und übersichtliche Menüs.
- Kontextbezogene Popup-Menüs in fast allen Bereichen des Programms. Dadurch wird es praktisch überflüssig, langwierige Befehle im Kopf zu behalten und einzugeben.

- Dialogfenster, in denen Sie Optionen setzen, umschalten und anschauen sowie Informationen eingeben können.

- Wenn Sie Text eingeben müssen, führt TDW eine Liste von allen Texten mit, die Sie in vergleichbaren Situationen geschrieben haben. Sie können aus diesen Eingaben wählen, den Text editieren oder neuen Text einfügen.

- Volle Makrosteuerung, um Befehlsfolgen und Eingaben zu beschleunigen.

- Kopieren und Einfügen zwischen Fenstern und Dialogfenstern.

- Bequeme und umfassende Fensterverwaltung.

- Mausunterstützung.

- Zugang zu verschiedenen Arten von Online-Hilfesystemen.

- Aufzeichnung des Programmablaufs mit der Möglichkeit, das Programm rückwärts ausführen zu können.

- Debuggen ist sowohl mit einem als auch mit zwei Bildschirmen möglich.

Im weiteren Verlauf dieses Kapitels werden diese Eigenschaften der Benutzeroberfläche von TDW ausführlich behandelt.

Menüs und Dialogfenster

Wie viele andere Produkte von Borland bietet TDW ein komfortables Menüsystem mit einer Menüleiste in der obersten Bildschirmzeile. Die Menüleiste ist immer zugänglich, außer wenn ein Dialogfenster aktiviert ist.

Allen Einträgen auf der Menüleiste sind eigene Untermenüs (*Pulldown*Menüs) zugeordnet. Mit Hilfe der Pulldown-Menüs können Sie

- einen Befehl aufrufen;

- ein *Popup*-Menü öffnen. Popup-Menüs erscheinen nach der Auswahl von Menüpunkten, hinter denen ein dreieckiger Pfeil steht.

- ein *Dialogfenster224ffnen*Dialogfenster erscheinen bei allen Menüpunkten, auf die eine Ellipse (...) folgt.

Bedienung der Menüs

Ins Menüsystem kommen Sie mit einer der folgenden vier Methoden:

- Drücken Sie F10 und bewegen Sie dann den Cursor mit den Pfeiltasten zum gewünschten Menüpunkt. Drücken Sie dann die Eingabetaste CR.

- Drücken Sie F10 und dann den ersten Buchstaben der Menübezeichnung (*Leertaste*, F, E, V, R, B, D, O, W, H).

- Drücken Sie die Taste Alt zusammen mit einem der Anfangsbuchstaben der Befehle in der Menüleiste (*Leertaste*, F, E, V, R, B, D, O, W, H), um das ausgewählte Befehlsmenü zu aktivieren. Alt F bringt Sie zum Beispiel stets ins Menü *File*. Das Menü *System* erhalten Sie mit Alt -*Leertaste*.

- Klicken Sie mit der Maus in den gewünschten Befehl der Menüleiste.

Innerhalb der Menüs können Sie sich wie folgt umherbewegen:

- Drücken Sie die Tasten ← und →, um vom einem Menü ins andere zu gelangen. (Wenn Sie sich zum Beispiel im *File*-Menü befinden, kommen Sie mit → in das *Edit*-Menü.)

- Verwenden Sie die Tasten ↑ und ↓, um die Befehle in einem Menü durchzugehen.

- Benützen Sie die Tasten Pos1 und Ende, um jeweils zum ersten und letzten Menüpunkt zu gelangen.

- Wählen Sie einen Menübefehl aus und drücken Sie CR, um in ein Untermenü (Popup-Menü) oder Dialogfenster zu gelangen.

- Klicken Sie mit der Maus auf einen Befehl, um ein Untermenü oder Dialogfenster zu aktivieren.

Verlassen wird das Menüsystem durch eine der folgenden Aktionen:

- Drücken Sie Esc, um ein Menü zu verlassen. Solange Sie sich nicht in der obersten Menüebene befinden, kommen Sie dadurch in das Menü, das unmittelbar zuvor aktiviert war.

- Drücken Sie innerhalb eines Pulldown-Menüs Esc und kehren Sie zum aktiven Fenster zurück.

- Mit F10 gelangen Sie aus jeder Menüebene in das unmittelbar zuvor aktivierte Fenster.

- Klicken Sie mit der Maus in das aktive Fenster, um das Menüsystem zu verlassen und zum aktiven Fenster zurückzukehren.

Einige Befehle der Hauptmenüs kann man über spezielle Tastenkombinationen ausführen, die auch als Hotkeys oder Tastenkürzel bezeichnet werden. Steht ein solcher Hotkey zur Verfügung, so erscheint das entsprechende Tastenkürzel rechts neben dem Menübefehl.

Die Abbildungen in Kapitel 13, »Die Befehlsübersicht«, zeigen den vollständigen Menübaum von TDW. In Tabelle 13.1 sind alle Hotkeys aufgelistet. Eine Übersicht über alle Befehle, die TDW zur Verfügung stellt, finden Sie ebenfalls in Kapitel 13.

Dialogfenster

Viele Befehlsoptionen von TDW werden in *Dialogfenstern* angegeben. Ein Dialogfenster kann aus folgenden Dialogelementen bestehen:

Tabelle 2.1 Die Elemente von Dialogfenster und deren Funktion

Dialogelement	Beschreibung
Aktionsschalter	Ein Aktionsschalter ist anhand eines unterlegten Texts (auf Monochrom-Bildschirm durch eine inverse Darstellung) gekennzeichnet. Wenn Sie einen Aktionsschalter auswählen, wird die entsprechende Aktion sofort ausgeführt. So läßt sich ein Dialogfenster zum Beispiel durch die Auswahl des Schalters *OK* unter Übernahme der angegebenen Optionen verlassen (das Tastenkürzel für *OK* lautet Alt K), während die Einstellungen mit *Cancel* unter Abbruch des Dialogs verworfen werden können. Dialogfenster enthalten außerdem den Schalter *Help*, mit dem Sie online-Hilfestellungen erhalten.
Markierungsfeld	Ein Markierungsfeld ist ein Ein-/Ausschalter. Damit läßt sich eine Option ein- oder ausschalten. Wenn das Markierungsfeld eingeschaltet ist, erscheint ein *X* in eckigen Klammern: [X] .
Schaltfeld	In einem Schaltfeld kann man eine von mehreren Einstellungen vornehmen. Es kann immer nur eine Option ausgewählt werden. Die ausgewählte Option wird durch einen dicken Punkt in runden Klammern angezeigt.
Eingabefeld	In einem Eingabefeld können Strings (zum Beispiel der Name einer Datei) eingegeben werden. Eingabefelder sind häufig mit einer *History-Liste* verknüpft (siehe »Geschichtsstunden«).

Eine Liste enthält eine Reihe von Einträgen, aus der Sie einen Eintrag auswählen können (zum Beispiel eine von mehreren Dateien).

Mit Hilfe der Tasten Tab und Umschalt Tab können Sie sich in Dialogfenstern bewegen. Innerhalb von Auswahlfeldern verwenden Sie die Pfeiltasten. Innerhalb von Schaltfeldern verwenden Sie die Pfeiltasten, um die Einstellung zu verändern. Zum Umschalten eines Markierungsfelds wählen Sie diesen mit Tab aus und drücken CR.

Wenn Sie mit einer Maus arbeiten, ist die Bedienung eines Dialogfensters noch einfacher. Sie brauchen bloß das gewünschte Dialogelement anzuklicken. Um das Dialogfenster zu schließen, klicken Sie in das Schließsymbol in der linken oberen Ecke.

Hinweis: Sie können die Dialogelemente in einem Dialogfenster auch über deren Hotkeys auswählen. Der Hotkey ist als hervorgehobener Buchstabe des Befehls zu erkennen.

Gewußt, wo

Zusätzlich zum praktischen Menüsystem von Borland bietet TDW den Vorteil größerer Übersichtlichkeit und verringertem Lernaufwand dadurch, daß sich die Anzahl der Menüs in Grenzen hält.

Um diesen Punkt zu verstehen, müssen Sie sich zunächst klarmachen, daß TDW ein Programm ist, das kontextbezogen arbeitet. TDW ist stets genau auf dem laufenden darüber, welches Fenster Sie geöffnet haben, welcher Text ausgewählt ist und in welchem Teil des Fensters sich der Cursor befindet. Mit anderen Worten, der Debugger weiß genau, was Sie gerade anschauen und wo der Cursor ist, wenn Sie einen Befehl auswählen. Diese Informationen werden benutzt, wenn der Debugger auf Ihren Befehl reagiert. Folgendes Beispiel erläutert diesen grundlegenden Aspekt. Nehmen wir an, in Ihrem Programm steht folgende Zeile:

```
Zaehler[Grad] := Zaehler[Grad] + 1;
```

Wie Sie bei der Arbeit mit TDW feststellen werden, ist es leicht, Informationen über Datenstrukturen zu bekommen. Sie drücken einfach Strg I , um die Daten zu untersuchen. Wenn der Cursor auf Zaehler steht, informiert Sie TDW über den Inhalt der gesamten Array-Variablen. Wenn Sie aber den gesamten Array-Namen zusammen mit dem Index auswählen (d.h. hervorheben) würden und dann Strg I drücken, so würde Ihnen TDW nur ein Element zeigen.

Auf diese Art können Sie sich in immer feinere Programmdetails vortasten. Wenn Sie ein Array untersuchen, gelangen Sie über Strg I zur Ansicht eines bestimmten Elements.

Diese Art der Berücksichtigung des Kontexts gewährleistet eine extrem einfache Bedienung von TDW. Sie erspart Ihnen die Mühe, lange Reihen von Menübefehlen oder obskuren Kommandozeilenoptionen im Gedächtnis zu behalten und einzugeben. Sie gehen einfach zu dem Datenobjekt, das Sie untersuchen wollen oder wählen es mit Hilfe der Einfg -Taste oder der Maus aus. Dann rufen Sie den Befehl, zum Beispiel Strg I für *Inspect*, auf. TDW leistet immer sein Bestes, um Sie so gut wie möglich zu bedienen.

Die Kontextbezogenheit, die dem Benutzer das Leben erleichtert, erschwert jedoch die Dokumentation der Befehle. Dies ergibt sich aus der Tatsache, daß zum Beispiel Strg I im TDW mehrere Funktionen haben kann. Was ein Befehl bewirkt, hängt davon ab, wo der Cursor steht und welcher Text ausgewählt wurde.

Lokale Menüs

Ein anderer Aspekt der Kontextsensibilität liegt in der Verwendung von Popup-Menüs, die der Situation entsprechend erscheinen.

Popup-Menüs in TDW nennt man ortsabhängig, weil sie auf den speziellen Punkt zugeschnitten sind, der durch die Position des Cursors bezeichnet ist. Es ist wichtig, (ortsabhängige) Popup-Menüs und allgemeine Menüs auseinanderzuhalten. Hier sehen Sie nun einen Bildschirm, der sich aus beiden Menüarten zusammensetzt. (Sie können einen solchen Bildschirm freilich bei der Arbeit mit TDW nicht erzeugen, da beide Menüarten niemals gleichzeitig aktiv sein können.) Man unterscheidet zwischen allgemeinen Menüs und lokalen kontextbezogenen Menüs.

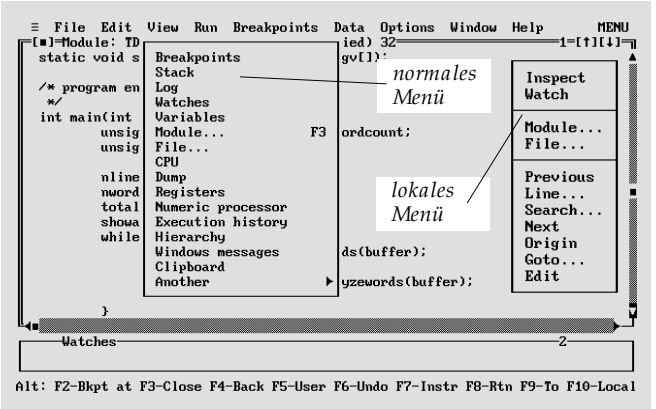


Abbildung 2.1 Menüs der Menüleiste im Vergleich mit Popup-Menüs

Allgemeine Menüs in der Menüleiste

■ Allgemeine Menüs sind diejenigen, zu denen man durch Betätigung der F10 -Taste und der Verwendung der Pfeiltasten beziehungsweise durch Eingabe des ersten Buchstaben des Menünamens gelangt.

■ Allgemeine Menüs sind immer verfügbar und auf der Menüleiste im oberen Bildschirmteil sichtbar.

■ Ihr Inhalt bleibt immer gleich.

■ Einige der Menübefehle haben Hotkey-Abkürzungen, die TDW in jeder Situation zur Verfügung stellt.

Lokale kontextbezogene Menüs

■ Ein Popup-Menü wird aufgerufen durch Eingabe von Alt F10 oder Strg F10 . Am bequemsten geht der Aufruf des lokalen Menüs durch Anklicken und Halten der rechten Maustaste.

■ Ort und Inhalt dieses Menüs hängen von der Textauswahl und der Cursorposition ab.

■ Die Inhalte von Popup-Menüs können sich ändern. (Auch wenn dies der Fall ist, sollte man sich merken, daß viele der lokalen Befehle in fast allen lokalen Menüs existieren. Ein bestimmter Grundstock an Befehlen ist also immer vorhanden.) Sogar die Auswirkungen von gleichnamigen Befehlen können je nach Kontext verschieden sein.

■ Bei einem Popup-Menü gibt es für jeden Befehl eine Abkürzung, bestehend aus Strg und dem ersten Buchstaben des Befehls. Dies bewirkt, daß ein Hotkey, zum Beispiel Strg S , in verschiedenen Situationen eine unterschiedliche Bedeutung hat. (Wie oben erwähnt, haben aber alle Popup-Menüs einen Kern von unveränderlichen Grundbefehlen. So führen zum Beispiel der Befehl *Goto* und der Befehl *Search* immer zum gleichen Ergebnis, auch wenn sie von verschiedenen Fensterausschnitten aufgerufen werden.)

Aus der Sicht des Anwenders sind lokale Menüs eine sehr große Erleichterung. Alle möglichen Befehle sind mit einem Blick zu erfassen. Damit wird die Eingabe falscher Befehle verhindert. Außerdem bleiben die Menüs klein und übersichtlich.

Geschichtsstunden

Menüs und kontextbezogene Befehle sind nur zwei Gesichtspunkte der Benutzeroberfläche von TDW. Eine weitere Besonderheit, an die man sich schnell gewöhnt, ist die Verfügbarkeit bereits erfolgter Eingaben in einer History-Liste.

Ausgehend von der Philosophie, daß der Benutzer nicht mehr Eingaben als unbedingt nötig machen soll, speichert TDW alle Eingaben in den Dialogfenstern und stellt den Text der letzten Eingabe bei einem erneuten Aufruf der Dialogfenster sofort wieder zur Verfügung.

Wenn Sie zum Beispiel die Funktion `Investitionsgewinn` suchen, dann müßten Sie normalerweise das ganze Wort oder einen Teil des Worts eingeben. Nehmen wir weiter an, daß Sie anschließend eine Variable mit der Bezeichnung `MeinAnteil` suchen. Wenn diesmal das Dialogfenster erscheint, sehen Sie den Text `Investitionsgewinn` im Dialogfenster. Wenn Sie nun einen weiteren Text suchen, bleiben die beiden Stringeingaben im Dialogfenster erhalten. Mit weiteren Suchbefehlen verlängert sich die Liste im Dialogfenster.

Das Dialogfenster zum Suchen sieht dann vielleicht wie folgt aus:

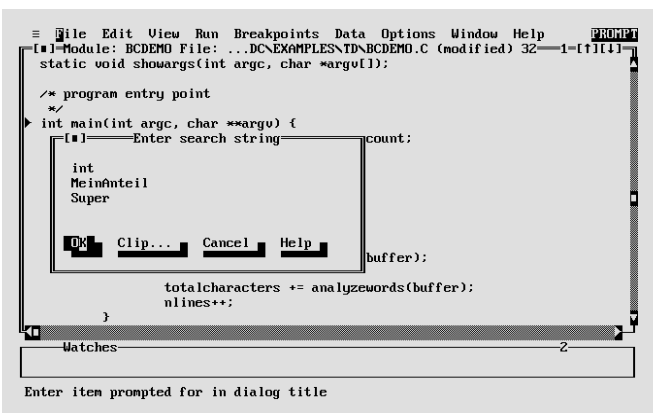


Abbildung 2.2 Eine History-Liste in einem Dialogfenster

Mit Hilfe der History-Liste können Sie die Eingabe für einen Suchvorgang wesentlich beschleunigen. Um die Suche zu starten, brauchen Sie nur noch einen zuvor eingegebenen Text mit den Pfeiltasten selektieren und dann die Eingabetaste CR zu drücken. Wenn Sie einen unveränderten Eintrag aus der History-Liste wiederverwenden, dann finden Sie diesen Eintrag beim erneuten Aufruf der History-Liste an der ersten Stelle. Der Text kann auch editiert werden (man verwendet dazu die Pfeiltasten, um den Cursor im markierten Text umher

zu bewegen, und nimmt zum Löschen wie üblich Entf oder Rücktaste). Sie können zum Beispiel `MeinAnteil` auswählen und in `SeinAnteil` umwandeln, anstatt den ganzen Text neu einzugeben. Wenn Sie einen neuen Text in einem hervorgehobenen Eintrag eingeben, dann wird der hervorgehobene Eintrag überschrieben.

TDW speichert, falls nicht anders konfiguriert, die letzten zehn Eingaben. (Sie können den Umfang an Einträgen mit TDWINST verändern, siehe Anhang C, »Die Konfiguration von Turbo Debugger«.)

Für die meisten Dialogfenster hält TDW eine eigene History-Liste bereit. Daher kann es nicht passieren, daß man Suchtexte, wie Zeilennummern oder bestimmte Labels, wegen der vielen Einträge nicht mehr findet.

Automatische Ergänzung des Namens

Es bietet sich immer, wenn Sie nach einem Eintrag für ein Eingabefeld gefragt werden, die Möglichkeit, nur einen Teil der Symbolbezeichnung einzugeben und dann Strg N zu drücken.

Achtung: Wenn die Meldung `READY...` gefolgt von drei Punkten in der rechten oberen Ecke des Bildschirms erscheint, heißt dies, daß die Symboltabelle gerade sortiert wird. Strg N funktioniert nicht, solange die drei Punkte am Bildschirm stehen und die Symboltabelle noch in Bearbeitung ist.

- Wenn Sie so viele Anfangsbuchstaben eingegeben haben, daß der Name eindeutig ermittelt werden kann, wird der Namen von TDW automatisch ergänzt.

- Wenn der eingegebene Name nicht mit dem Anfang einer Symbolbezeichnung übereinstimmt, passiert gar nichts.

- Wenn durch die Eingabe mehr als ein Symbol gemeint sein könnte, erscheint eine Liste mit den übereinstimmenden Bezeichnungen, aus der Sie die gewünschte auswählen können.

Sukzessive Auswahl

In TDW gibt es außerdem die Möglichkeit, die Einträge in einem Dialogfenster nach und nach durch die Eingabe der Anfangsbuchstaben auszuwählen. Wenn der Name in der Liste vorkommt, wird der Auswahlbalken auf die erste Bezeichnung in der Liste gesetzt, die sich mit den Anfangsbuchstaben deckt. Hat man die gewünschte Bezeichnung gefunden, so kann sie mit *OK* ausgewählt werden.

Makros erstellen

Makros sind einfach Tastenkürzel für Tastenfolgen, die Sie frei definieren können. Sie können damit eine Reihe von Turbo-Debugger-Befehlen und Tasta-

tureingaben auf einen einzigen Tastendruck legen und diese somit jederzeit abrufbar machen. (Jedesmal, wenn Sie eine Reihe von Schritten wiederholen, sollten Sie sich fragen, ob Sie nicht ein Makro für diesen Ablauf erstellen wollen.)

In Kapitel 4, »Der Aufruf von TDW«, ist beschrieben, wie Makros definiert und verwendet werden können.

Schaufenster-bummel

TDW stellt alle Informationen und Daten in (lokalen und globalen) Menüs, in Dialogfenstern (für Eingaben) und in Fenstern dar. Es gibt viele verschiedene Fenster, je nachdem, welche Art Information dargestellt wird. Sie öffnen und schließen alle Fenster mit Hilfe von Menübefehlen (oder den Tastenkürzeln für diese Befehle). Die meisten Fenster von TDW werden im *View*-Menü geöffnet (dort stehen fünfzehn Fenster zur Auswahl). Eine weitere Art von Fenstern wird als *Inspector*-Fenster bezeichnet. Solche Fenster öffnet man mit Hilfe des Befehls *Data/Inspect* oder durch Aufruf des Befehls *Inspect* in einem lokalen Menü.

Fenster des View-Menüs

Es gibt fünfzehn Fenster, die im *View*-Menü aufgerufen werden können. Ein geöffnetes Fenster läßt sich mit den Befehlen der Menüs *Window* und *System* verschieben, vergrößern, verkleinern und schließen usw. Diese Menüs werden im Abschnitt »Die Arbeit mit Fenstern« beschrieben.

Module stellt den Programmcode dar, den Sie debuggen. Sie können sich im ganzen Modul frei bewegen und Daten und Code untersuchen, indem Sie auf die Programmvariablen mit dem Cursor zeigen und den entsprechenden Befehl des lokalen Menüs eingeben.

Sie werden wahrscheinlich mehr Zeit im *Module*-Fenster als in allen anderen Fenstern verbringen. Deshalb sollten Sie sich ausreichend Zeit zum Erlernen der verschiedenen lokalen Menübefehle dieses Fensters nehmen.

Sie können ein *Module*-Fenster auch mit Alt F3 öffnen. (In Kapitel 8, »Dateien untersuchen und ändern«, werden das *Module*-Fenster und dessen Befehle ausführlich erläutert.)

Watches zeigt Variablen und wie sich deren Werte verändern. Sie können eine Variable in diesem Fenster hinzufügen, indem Sie den Cursor auf eine Variablenbezeichnung setzen und Strg W eingeben. (Nähere Informationen über das *Watches-Fenster* erhalten Sie in Kapitel 6, »Programmdaten untersuchen und ändern«).

Breakpoints enthält die von Ihnen gesetzten Breakpoints. Ein Breakpoint bezeichnet eine Stelle in Ihrem Programm, an der eine bestimmte Aktion vorgenommen werden soll, zum Beispiel das Anhalten des Programms, damit Sie den Zustand des Programms überprüfen können. Der linke Ausschnitt zeigt die Positionen der einzelnen Breakpoints (oder gibt an, daß der Breakpoint global ist). Im rechten Ausschnitt stehen die Bedingungen, unter denen der

gerade ausgewählte Breakpoint ausgelöst wird. Verwenden Sie dieses Fenster, um Breakpoints zu verändern, zu löschen oder hinzuzufügen. (In Kapitel 7, »Breakpoints«, finden Sie eine vollständige Besprechung dieses Fensters und der Funktionsweise der Breakpoints.)

Stack stellt den aktuellen Zustand des Stack dar, wobei die erste aufgerufene Funktion (in C ist das `main`) unten steht und alle nachfolgend aufgerufenen Funktionen der Reihe nach darüber stehen.

Sie können weitere Informationen über die einzelnen Funktionen oder Prozeduren erhalten, indem Sie den Cursor auf eine solche setzen und diese mit Strg I untersuchen (*Inspect*).

Wenn man den Cursor auf eine der Funktionen oder Prozeduren in der Liste bewegt und Strg L eingibt, kann man entweder die lokalen Variablen oder die Argumente, mit denen die Routine aufgerufen wurde, untersuchen. (Kapitel 5, »Steuerung der Programmausführung«, enthält weitere Informationen über das *Stack*-Fenster.)

Log enthält ein Protokoll aller Meldungen. Es speichert eine Liste an Mitteilungen und Informationen, die Sie während der Arbeit mit TDW sammeln können (Informationen, warum das Programm angehalten wurde, die Ergebnisse von Breakpoints, die Werte von Strukturen, usw.).

Mit diesem Fenster können Sie auch Informationen über die Speicherbenutzung und Module einer Microsoft Windows-Anwendung erhalten.

Dieses Fenster ermöglicht Ihnen, zurückzublicken und nachzuvollziehen, warum es zu diesem oder jenem Ereignis kommen konnte. Diese Funktion werden Sie sehr schnell zu schätzen wissen, beispielsweise, wenn Sie sich in einem Programm länger umgesehen haben und sich am Ende fragen, wie Sie wohl hierher gelangt sind. (Kapitel 7, »Breakpoints«, enthält weitere Informationen über das *Log*-Fenster.)

Variables enthält alle Variablen, die momentan von Ihrem Programm aus verwendet werden können. Der obere Ausschnitt enthält die globalen, der untere die lokalen Variablen.

Dieses Fenster ist hilfreich zum Auffinden bestimmter Funktionen oder Variablen, deren Namen man nicht mehr genau weiß. Sie können im globalen Variablenausschnitt nachsehen und so den Namen schnell herausfinden. (In Kapitel 5, »Steuerung der Programmausführung«, wird das Variablen-Fenster ausführlich beschrieben.)

File zeigt den Inhalt einer Datei. Die Datei kann entweder hexadezimal oder als ASCII-Text dargestellt werden. Sie können Strings oder Bytefolgen suchen oder direkt einen bestimmten Teil einer Datei auf einer Diskette patchen.

Diese Möglichkeit empfiehlt sich, wenn Sie ein Programm debuggen, das Dateien verwendet, deren Änderung eine Änderung des Programmverhaltens bewirken sollen. Sie können auch Fehler in Dateien korrigieren oder eine Datei, die vom Programm erzeugt wird, untersuchen, um nachzuprüfen, ob ihr Inhalt korrekt ist. (In Kapitel 8, »Dateien untersuchen und ändern«, erfahren Sie mehr über das *File*-Fenster.)

CPU gibt den aktuellen Zustand der CPU wieder. Dieses Fenster besteht aus sechs Ausschnitten: ein Ausschnitt enthält den disassemblierten Programmcode, einer den Inhalt eines Selectors, ein weiterer zeigt einen Datenbereich hexadezimal an, ein anderer enthält den Stack in Form von Hexadezimalworten, im fünften sehen Sie den Inhalt der CPU-Register und im letzten den Status der CPU-Flags.

Das *CPU*-Fenster werden Sie für die Untersuchung von Maschinensprachebefehlen einer Quellcodezeile oder einzelner Byte einer Datenstruktur benutzen. Wenn Sie den Assemblercode verstehen, können Sie damit auch sehr vertrackte Fehler finden. Für die Mehrzahl der Programme benötigt man dieses Fenster nicht. (In Kapitel 11, »Debuggen auf Assemblerebene«, wird das *CPU*-Fenster näher erläutert.)

CPU-Fenster werden von TDW manchmal automatisch geöffnet, wenn das Programm bei einem Befehl angehalten wird, der sich zwischen zwei Quellcodezeilen befindet.

Dump zeigt einen Speicherbereich in der reinen Datendarstellung. Sie können die Daten als Zeichen, Hex-Byte, Worte, Doppelworte und in allen Fließkommaformaten darstellen. (Dieses Fenster entspricht dem Datenausschnitt des *CPU*-Fensters.) Sie können dieses Fenster verwenden, wenn Sie einen Datenbereich, nicht aber den CPU-Zustand darstellen wollen. Das lokale Menü enthält viele Befehle, mit denen Sie die dargestellten Daten und die Darstellungsart verändern können. Sie können auch ganze Speicherblöcke manipulieren und direkt auf die Ein-/Ausgabe-Ports zugreifen. (In Kapitel 11, »Debuggen auf Assemblerebene«, erhalten Sie weitere Informationen zur Verwendung dieses Fensters.)

Registers enthält die CPU-Register und die Flags. Dieses Fenster hat zwei Ausschnitte, die den Register- und Statusausschnitten des CPU-Fensters entsprechen. Verwenden Sie dieses Fenster, wenn Sie die Register sehen wollen, aber auf den Rest des CPU-Fensters verzichten können. Mit Hilfe der Befehle des lokalen Menüs können Sie die Werte der Register und Flags beliebig manipulieren. (In Kapitel 12, »Debuggen auf Assemblerebene«, erhalten Sie weitere Informationen zur Verwendung dieses Fensters.)

Numeric processor stellt den aktuellen Zustand des Koprozessors dar. Dieses Fenster besteht aus drei Ausschnitten: ein Ausschnitt enthält die Fließkommaregister, ein weiterer Ausschnitt zeigt die Werte der Statusbits und ein anderer Ausschnitt stellt die Kontrollbits dar.

Dieses Fenster unterstützt die Diagnose von Programmen mit Fließkommazahlen. Sie müssen ein gutes Verständnis für die Arbeitsweise des Koprozessors mitbringen, wenn Sie von diesem Fenster wirklich profitieren wollen. (In Kapitel 12, »80x87-Koprozessor und Fließkomma-Emulator«, finden Sie weitere Informationen über das Fenster *Numeric processor*.)

Execution history stellt die Quellcodezeilen Ihres Programms bis zur letzten bereits abgearbeiteten Zeile hin dar. Folgende Informationen können Sie diesem Fenster entnehmen:

1. ob Sie das Programm schrittweise durchgehen;
2. die Nummer der Zeile im Quellcode, die zur Ausführung ansteht;
3. die Zeilennummer im Quellcode.

Sie können diese Fenster zu Ihrer Information heranziehen oder das Programm an eine bestimmte Stelle zurücklaufen lassen. (Weitere Informationen finden Sie in Kapitel 5, »Steuerung der Programmausführung«.)

Hierarchy enthält eine Liste mit allen Objekten oder Klassen des aktuellen Moduls und stellt deren Beziehungen in einem Hierarchiebaum dar. Das Fenster hat zwei Ausschnitte: einen für die Liste mit den Objekttypen und Klassen sowie einen für den Hierarchiebaum der Objekte/Klassen. (Wenn Sie ein Programm debuggen, das die Mehrfachvererbung von C++ verwendet, wird auch noch ein dritter Ausschnitt dargestellt, in dem die Vorfahren einer bestimmten Klasse wiedergegeben werden.)

Dieses Fenster veranschaulicht die Beziehungen zwischen den Objekten und Klassen des aktuellen Moduls. Damit ist es außerdem möglich, einen Objekttyp oder eine Klasse mit allen Ihren Datenfeldern und Methoden oder Datenelementen und Elementfunktionen über die Befehle des lokalen Menüs zu untersuchen. (In Kapitel 10, »Debuggen in C++ und objektorientiertem Pascal«, finden Sie weitere Erläuterungen zum *Hierarchy*-Fenster.)

Windows Messages In Kapitel 11, »TDW für Windows (TDW)«, wird erklärt, wie Windows Messages verwendet wird. Das Fenster *Windows Messages* zeigt eine Liste der Meldungen, die zwischen den Fenstern Ihrer MS-Windows-Anwendung übertragen wurden. Es besteht aus drei Ausschnitten:

- Im linken Ausschnitt wird angezeigt für welche Prozeduren oder Handles Sie die Meldungen aufzeichnen.
- Der rechte Ausschnitt zeigt den Meldungstyp.
- Im unteren Ausschnitt werden die Meldungen dargestellt.

Clipboard-Fenster
enthält die Objekte und deren Typ, die in die Zwischenablage übernommen wurden. Die Objekte können untersucht, gelöscht oder festgehalten werden (siehe nächsten Abschnitt, »Kopieren und Einfügen«, in diesem Kapitel).

Mehrfache Fenster lassen sich für drei Arten von Fenstern öffnen – *Dump*, *File* und *Module* – indem man den Befehl *View/Another* auswählt. Damit können Sie mehrere getrennte Bereiche des Assemblercodes, unterschiedliche Dateien und mehrere Programmbereiche auf einmal betrachten.

Sie können unbesorgt sein, wenn TDW eines dieser Fenster von selbst öffnet. Dies passiert als Reaktion auf manche Befehle.

Benutzerbildschirm

Der Ausgabebildschirm ist ein Fenster, das den gesamten Bildschirm mit den Ausgaben Ihres Programms zeigt. Der Bildschirm ist exakt derselbe wie der, den Sie erhalten, wenn Sie Ihr Programm von DOS und nicht von TDW aus starten.

Dieses Fenster ermöglicht Ihnen nachzuprüfen, ob Ihr Programm auch tatsächlich da ist, wo Sie es erwartet haben, und ob alle Bildschirmausgaben auch wie erwartet erfolgen. Das Tastenkürzel für dieses Fenster ist Alt F5 . Mit dem nächsten Tastendruck kommen Sie wieder zu TDW zurück.

Inspector-Fenster

Ein *Inspector*-Fenster stellt den aktuellen Inhalt der ausgewählten Variablen dar. Mit Hilfe des Befehls *Data/Inspect* oder dem Befehl *Inspect* eines lokalen Menüs wird das *Inspector*-Fenster geöffnet. Das Fenster wird normalerweise mit Esc oder durch einen Mausklick auf das Schließsymbol geschlossen. Wenn Sie mehr als ein *Inspector*-Fenster aufgemacht haben, was bei der Untersuchung komplexer Datenstrukturen häufig vorkommen kann, dann können Sie alle Fenster mit Alt F3 oder *Window/Close* auf einmal schließen.

Sie können ein *Inspector*-Fenster öffnen, um sich ein Array von Daten oder den Inhalt einer Variablen oder einen Ausdruck anzuschauen. Ein *Inspector*-Fenster paßt sich immer an den Datentyp der dargestellten Objekte an. Man kann damit einfache Skalare (`int`, `float` usw.) darstellen, aber auch Zeiger, Records, Strukturen und Varianten. Jeder Datentyp wird so dargestellt, daß die Darstellung der Daten so nah wie möglich an die Darstellung in Ihrer Quelldatei herankommt.

Beachten Sie bitte, daß Sie im Gegensatz zu den Fenstern des *View*-Menüs beliebig viele *Inspector*-Fenster mit Hilfe des Befehls *Inspect* erzeugen können. Weitere *Module*-, *File*- und *CPU*-Fenster kann man nur mit dem Befehl *View/Another* erzeugen.

Das aktive Fenster

Auch wenn Sie in TDW viele Fenster öffnen können, kann es nur *ein* aktives Fenster geben. Sie können das aktive Fenster anhand der folgenden Kriterien erkennen:

- Das aktive Fenster ist mit einer doppelten, nicht mit einer einfachen Linie eingerahmt.
- Der Titel des aktiven Fensters ist hervorgehoben.
- Der Cursor oder Auswahlbalken befindet sich immer im aktiven Fenster.
- Überlappen sich mehrere Fenster, so ist das oberste Fenster das aktive.

Wenn Sie Befehle auswählen, Texte eingeben, oder Informationen auf und ab rollen, ist ausschließlich das aktive Fenster betroffen. Alle anderen eventuell geöffneten Fenster bleiben unverändert.

Woraus besteht ein Fenster?

Ein Fenster ist ein Teil des Bildschirms und kann in der Regel manipuliert werden. Die meisten Fenster folgende Elemente gemeinsam:

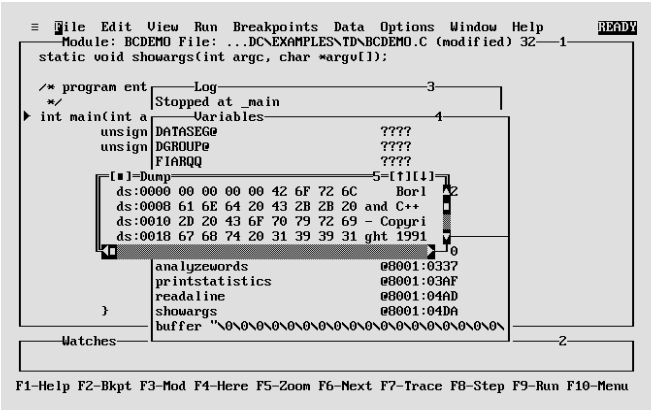


Abbildung 2.3 Können Sie das aktive Fenster erkennen?

- einen Rahmen (doppelt, wenn das Fenster aktiv, sonst einfach);
- einen Titel links oben;
- eine Bildlaufleiste rechts oder unten, wenn das Fenster weitere Informationen enthält, die nicht auf einmal dargestellt werden können. Die Bildlaufleisten können mit der Maus wie folgt bedient werden:
  - Klicken Sie auf die Richtungspfeile an den beiden Enden der Balken, um das Fenster eine Zeile oder ein Zeichen in die gewünschte Richtung zu verschieben.
  - Klicken Sie in den grau unterlegten Bereich in der Mitte des Balkens, um den Inhalt um eine Bildschirmseite in die gewünschte Richtung zu verschieben.
  - Ziehen Sie das Symbol für den Bildlauf an die gewünschte Position.
- ein Größenänderungsfeld in der unteren rechten Ecke. Ziehen Sie dieses Symbol mit der Maus an die gewünschte Position, um das Fenster zu vergrößern oder zu verkleinern. Wenn sich unten oder rechts an einem Fenster keine Bildlaufleisten befinden, kann die Größe des Fensters auch durch Verschieben dieser Seitenlinien geändert werden.

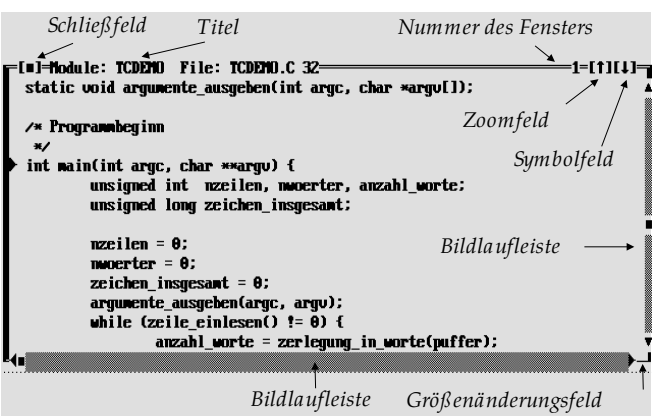


Abbildung 2.4 Ein typisches Fenster

- eine Fensternummer in der rechten oberen Ecke, mit der die Reihenfolge angegeben wird, in der das Fenster geöffnet wurde;

- ein Zoom- und ein Symbolfeld in der rechten oberen Ecke. Das linke Feld ist das Zoomfeld, das rechte ist das Symbolfeld. Klicken Sie mit der Maus in diese Symbole, um das Fenster auf volle Bildschirmgröße oder auf die ursprüngliche Größe zu bringen oder in ein Symbol umzuwandeln. (Wenn ein Fenster auf volle Bildschirmgröße ausgedehnt wird, ist nur noch das Symbolfeld verfügbar. Nach der Umwandlung in ein Symbol ist nur noch das Zoomfeld vorhanden.)

- ein Schließfeld in der linken oberen Ecke. Klicken Sie das Feld mit Ihrer Maus an, um das Fenster zu schließen.

Die Arbeit mit Fenstern

Angesichts der vielen verschiedenen Fenster werden Sie wahrscheinlich häufig mehrere Fenster gleichzeitig geöffnet haben. In TDW ist es nicht schwer, von einem Fenster in ein anderes zu springen, ein Fenster umherzubewegen, über ein anderes zu legen, zu verkleinern, falls es Sie bei der Arbeit stört, oder zu vergrößern, damit Sie besser damit arbeiten können, und zu schließen, falls die Bearbeitung abgeschlossen ist.

Die meisten Befehle zur Verwaltung der Fenster befinden sich im Menü *Windows*. Einige weitere Befehle stehen im System-Menü. Dieses Menü erkennen Sie an den drei waagerechten Linien auf der linken Seite der Menüleiste. Mit Alt -Leertaste können Sie das System-Menü öffnen, das Menü *Windows* läßt sich mit Alt W aktivieren.

Von Fenster zu Fenster wechseln Jedes Fenster ist in der rechten oberen Ecke mit einer Numerierung versehen. Normalerweise ist das *Module*-Fenster das Fenster 1 und das *Watches* -Fenster das Fenster 2. Was für ein Fenster Sie anschließend auch öffnen, spielt keine Rolle, es ist in jedem Fall das Fenster 3 usw.

Diese Numerierung ist die Grundlage für eine schnelle Methode, um von einem Fenster in ein anderes zu wechseln. Jedes der ersten neun geöffneten Fenster läßt sich aktivieren, indem man Alt zusammen mit der Nummer des Fensters, die in der rechten oberen Ecke steht, eingibt. Wenn Sie zum Beispiel Alt 2 eingeben, wird das *Watches* -Fenster aktiviert. Alle nachfolgenden Befehle beziehen sich nun nur auf dieses Fenster und die darin enthaltenen Einträge.

Mit einer Maus kann ein Fenster einfach durch Anklicken aktiviert werden.

Über das *Window* -Menü erhalten Sie eine Liste der geöffneten Fenster, aus der Sie ein Fenster auswählen können. Mit der Eingabetaste CR aktivieren Sie das jeweilige Fenster. (Sie können auch F6 betätigen, um von einem Fenster zum anderen zu wechseln.)

Wenn mehr als neun Fenster geöffnet sind, können Sie trotzdem ein Fenster mit *Window Pick* auswählen; wählen Sie diesen Befehl aus, um sich das gewünschte Fenster in einer Popup-Liste auszusuchen.

Wenn ein Fenster aus einzelnen Ausschnitten – das sind Bereiche in einem Fenster, die für bestimmte Daten verwendet werden – besteht, können Sie mit

Tab oder Umschalt Tab von einem Ausschnitt zum anderen wechseln. Sie können aber auch den Befehl *Window/Next pane* ausführen.

Sie können den gewünschten Ausschnitt auch einfach mit der Maus anklikken.

Das *CPU*-Fenster hat in TDW die meisten (sechs) Ausschnitte.

Wenn Sie von Ausschnitt zu Ausschnitt wechseln, sehen Sie vielleicht manchmal einen blinkenden Cursor, während in manchen Ausschnitten ein Auswahlbalken erscheint. Erscheint ein Cursor, können Sie sich mit den Cursortasten im Text umherbewegen. (Mit Bild↑, Strg Pos1 und Strg Bild↓ bewegen Sie sich zum Beispiel entweder eine Bildschirmseite nach oben oder in die oberste Zeile des Fensterausschnitts beziehungsweise an die oberste Position in der Liste.) Sie können auch die Wordstar-Befehle für die Cursorbewegungen in einem Fensterausschnitt verwenden. In Kapitel 13, »Die Befehlsübersicht«, finden Sie tabellarische Zusammenfassungen der Tastaturbefehle für die einzelnen Fensterausschnitte.

Erscheint in einem Fenster anstelle eines Cursors ein Auswahlbalken, kann man ebenfalls die Cursortasten verwenden, um einen Eintrag auszuwählen. Es gibt jedoch auch hier eine Reihe von speziellen Tastaturbefehlen. In alphabetisch sortierten Listen kann man zum Beispiel direkt über Tastatureingabe auswählen. Bei jedem eingegebenen Buchstaben springt der Auswahlbalken auf das erste Objekt, das mit dem soeben eingegebenen Buchstaben anfängt. Die Cursorposition in der hervorgehobenen Zeile deutet an, wieviele Zeichen eines Namens bereits eingegeben wurden. Wenn sich der Auswahlbalken über dem gewünschten Objekt befindet, ist die Suche abgeschlossen. Die Suche, die Schritt für Schritt vor sich geht, verringert die Anzahl der Zeichen, die Sie sonst für die Auswahl eines Objekts in einer Liste eingeben müßten.

Sobald ein Objekt in einer Liste ausgewählt ist, können Sie mit Alt F10 einen lokalen Befehl aufrufen. In vielen Listen braucht man auch nur die Eingabetaste CR zu betätigen, wenn das Objekt selektiert ist. Das ist meist die Kurzform für einen der häufig verwendeten Befehle eines lokalen Menüs. Die jeweilige Funktion der Eingabetaste CR in den einzelnen Fensterausschnitten ist im Referenzabschnitt in Kapitel 13, »Die Befehlsübersicht«, beschrieben.

Viele Fensterausschnitte ermöglichen die Eingabe neuer Werte oder Strings, ohne daß vorher ein Befehl ausgewählt wurde. Diese Möglichkeit besteht nur für den am häufigsten verwendeten Befehl in einem Fensterausschnitt oder einem Fenster (zum Beispiel für *Goto* in einem *Module*-Fenster, für *Search* in einem *File*-Fenster oder für *Change* in einem *Registers*-Fenster).

Fenster anpassen und verschieben Wenn TDW ein neues Fenster öffnet, erscheint das Fenster in der Nähe der aktuellen Cursorposition. Es erhält eine voreingestellte Größe, die den Anforderungen des Fensters entspricht. Mit Hilfe des Befehls *Window/Size/Move* kann man die Größe und die Position des Fensters verändern. Nach Auswahl dieses Befehls wird der Rahmen des aktiven Fensters mit einer einfachen Linie dargestellt. Sie können dann die Pfeiltasten bzw. die Taste Umschalt in Verbindung mit den Pfeiltasten verwenden, um die Position bzw. die Größe des Fensters auf Ihrem Bildschirm festzulegen. Sind Sie mit der Einstellung zufrieden, drücken Sie die Eingabetaste CR (Sie können auch das Tastenkürzel Strg F5 anstelle von *Window/Size* verwenden.)

Mit einer Maus ist das Verschieben und die Größenveränderung eines Fensters besonders einfach:

- Ziehen Sie das Vergrößerungssymbol in der rechten untern Ecke an die gewünschte Position.

- Ziehen Sie an der Überschrift des Fensters oder den Kanten des Fensters (aber nicht die Bildlaufleisten), um das Fenster umherzubewegen.

Falls Sie ein Fenster kurzfristig vergrößern (oder verkleinern) wollen, können Sie dies mit dem Befehl *Window/Zoom* oder einem Mausklick in das Symbol in der rechten oberen Ecke bewerkstelligen. Das Tastenkürzel für *Window/Zoom* ist F5.

Wenn Sie das Fenster nur vorübergehend entfernen, aber nicht schließen wollen, können Sie das Fenster aktiv machen und anschließend *Windows/Iconize/Restore* auswählen. Das Fenster wird daraufhin in ein kleines Symbol verwandelt, das nur noch den Namen, das Schließfeld und das Zoomfeld enthält. Um das Fenster in die ursprüngliche Form zurückzuverwandeln, aktivieren Sie das Symbol und wählen erneut *Window/Iconize/Restore* aus oder klicken mit der Maus in das Zoomfeld.

Fenster schließen und wiederherstellen Wenn Sie die Arbeit an einem Fenster beendet haben, können Sie das Fenster mit *Window/Close* oder Alt F3 schließen.

Wenn Sie mit einer Maus arbeiten, reicht es, in das Schließfeld in der linken oberen Ecke des Fensters zu klicken.

Falls Sie ein Fenster versehentlich schließen, können Sie das Fenster mit *Window/Undo close* oder Alt F6 wiederherstellen. Das funktioniert aber nur mit dem zuletzt geschlossenen Fenster.

Sie haben die Möglichkeit, den Bildschirm von TDW wieder so herzustellen, als Sie das Programm aufgerufen haben. Wählen Sie dazu den Befehl *System/Restore standard* aus.

Falls Ihr Bildschirm durch die Ausgaben Ihres Programms zerstört wurde (weil Sie vielleicht die Bildschirmzwischenspeicherung abgeschaltet haben), können Sie die Darstellung mit dem Befehl *System/Repaint desktop* wiederherstellen.

Speichern der Fensteranordnung Mit Hilfe des Befehls *Options/Save* kann man eine bestimmte Fenstereinstellung abspeichern. Im Dialogfenster *Save configuration* können Sie das Auswahlfeld *Layout* angeben. Der Bildschirm erscheint beim Aufruf von TDW dann immer in dieser Anordnung, falls Sie Ihre Konfiguration in einer Datei mit dem Namen TDCONFIG.TDW abgespeichert haben. Diese Datei ist die einzige, die von TDW automatisch beim Start geladen wird. Andere in verschiedenen Dateien gespeicherte Konfigurationen können mit Hilfe des Hauptmenübefehls *Options/Restore options* oder auch beim Start des Debuggers durch Angabe des Kommandozeilenparameters *-c* geladen werden, wenn sie unter einem anderen Dateinamen abgespeichert wurden.

Einfügen und Kopieren

TDW verfügt über eine sehr praktische Möglichkeit zum Kopieren und Einfügen von Daten: die Zwischenablage. Mit Hilfe der Zwischenablage kann man in Turbo Debugger problemlos Daten zwischen einzelnen Fenstern und Dialogfenstern austauschen und übertragen.

Der in die Zwischenablage kopierte Inhalt wird dynamisch verwaltet; wenn sich der Wert eines Eintrags in Ihrem Programm ändert, so ändert sich auch automatisch der Wert des Eintrags in der Zwischenablage.

Um einen Eintrag in die Zwischenablage zu übernehmen, stellen Sie den Cursor über den entsprechenden Eintrag oder heben ihn mit der Taste Einfg hervor und drücken dann Umschalt F3. Zum Einfügen eines Eintrags aus der Zwischenablage in ein Fenster oder ein Dialogfenster drücken Sie Umschalt F4, oder Sie verwenden den Schalter *Clip* im Dialogfenster, um das Dialogfenster der Zwischenablage zu öffnen.

Sie können den Eintrag der Zwischenablage in jedes beliebige Eingabefeld eines Dialogfensters einfügen, indem Sie Umschalt F4 drücken, auch wenn das Dialogfenster keinen *Clip*-Schalter enthält. Es ist zudem möglich, den Inhalt in ein Dialogfenster mit mehreren Feldern einzufügen.

Das Dialogfenster Pick

Mit Umschalt F4 oder durch Betätigen des Schalters *Clip* öffnen Sie ein Dialogfenster, das die Einträge der Zwischenablage und die Kategorien zeigt, die darüber bestimmen, auf welche Art ein Eintrag eingefügt werden kann.

Dieses Dialogfenster zeigt die Liste der Einträge, die sich in der Zwischenablage befinden. Die Einträge können auf drei verschiedene Arten interpretiert werden: als String, als Adresse oder als Inhalt einer Adresse. Die Kategorien, die Sie zum Einfügen des Eintrags verwenden können, hängen von dessen Typ und dessen Verwendungszweck ab (wird später näher erläutert).

Wenn Sie zum Beispiel Text aus dem *Log*-Fenster kopieren, kann dieser nur als String eingefügt werden. Wenn Sie Text aus dem *Module*-Fenster überneh-

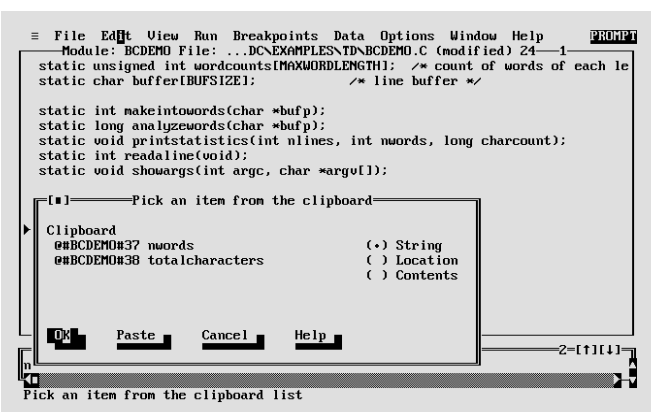


Abbildung 2.5 Das Dialogfenster Pick

men, kann er woanders als String oder Adresse eingfügt werden, aber nicht als Inhalt einer Adresse. Wenn Sie eine Variable aus einem *Inspector*-Fenster kopieren, kann diese als String, Adresse oder als Inhalt einer Adresse eingefügt werden (außer es handelt sich um eine Registervariable in C, die nur als String oder als Inhalt einer Adresse verwendet werden kann).

Um einen Eintrag in ein Dialogfenster einzufügen, heben Sie diesen Eintrag hervor, wählen die gewünschte Kategorie aus und drücken dann entweder die Eingabetaste oder den Schalter *Paste*, je nachdem, was Sie in das Dialogfenster übernehmen wollen.

- Mit der Eingabetaste wird der Eintrag eingfügt, und Sie kehren wieder in das Dialogfenster zurück.

- Der Schalter *Paste* bewirkt zweierlei: der markierte Eintrag wird in ein Eingabefeld des Dialogfensters eingefügt, gleichzeitig wird diese Eingabe bestätigt – die Funktion des Dialogfensters wird also ausgeführt.

Das Clipboard-Fenster

TDW stellt Ihnen für die Arbeit mit der Zwischenablage ein weiteres Fenster, ein *View*-Fenster, zur Verfügung, in dem Sie sich den Inhalt der Zwischenablage anzeigen lassen können. Dieses *Clipboard* -Fenster wird mit *View/Clipboard* geöffnet.

Im äußersten linken Feld dieses Fenster sehen Sie eine Beschreibung des Eintrags, auf die ein Doppelpunkt und der Eintrag selbst folgen. Wenn es sich um einen Ausdruck aus einem *Watches* -Fenster, eine Variable aus einem *Inspector*-Fenster oder Daten, Register oder Flags aus dem *CPU*-Fenster handelt, folgt stets der entsprechende Wert des Eintrags.

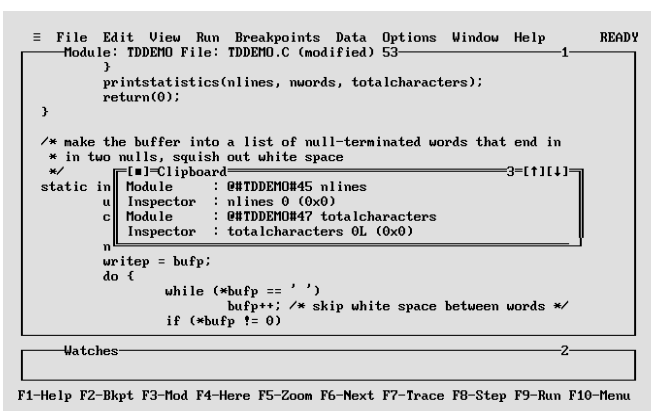


Abbildung 2.6 Das Dialogfenster Clipboard

Arten von Clipboard-Einträgen Wenn Sie einen Eintrag aus einem Fenster kopieren, weist TDW diesem Eintrag eine bestimmte Kategorie zu, damit Sie den Ursprung des Eintrags leichter erkennen können. Die folgende Tabelle zeigt die einzelnen Kategorien in der Zwischenablage:

Tabelle 2.2 Die verschiedenen Kategorien von Einträgen in der Zwischenablage

Typ	Beschreibung
String	Ein String, wie ein markierter Block im *File*-Fenster.
Module	Ein Modulkontext, mit einer Quelltextposition, wie bei einer Variablen des *Module*-Fensters.
File	Eine Position in einer Datei (im *File*-Fenster), die kein Programmmodul ist.
CPU code	Eine Adresse und eine Byte-Liste ausführbarer Befehle aus dem Code-Ausschnitt des CPU-Fensters.
CPU data	Eine Adresse und eine Byte-Liste von Daten aus dem Datenausschnitt des CPU-Fensters oder des *Dump*-Fensters.
CPU stack	Eine Quelltextpositition und ein Stack-Frame aus dem Stack-Ausschnitt des CPU-Fensters.
CPU register	*Eine Registerbezeichnung und ein Wert aus dem Register-Ausschnitt des CPU-Fensters oder des Register-Fensters.*
CPU flag	Ein Wert des Stautsregisters aus dem Statusausschnitt des CPU-Fensters.
Inspector	Einer der folgenden Werte: Ein Variablenbezeichner aus einem *Inspector*-Fenster. Ein konstanter Wert aus einem *Inspector*- oder *Watches-Fenster. Der Wert einer Registervariablen aus einem Inspector*-Fenster. Ein Bit-Feld aus einem *Inspector*-Fenster.
Address	Eine Adresse ohne zugehörige Daten- oder Code-Angabe.
Expression	Ein Ausdruck aus dem *Watches*-Fenster.
Coprocessor	Ein 80x87-Register des Koprozessors.
Control flag	Ein 80x87-Control-Flag.
Status flag	Ein 80x87-Status-Flag.

Das lokale Menü des Clipboard-Fensters Wenn Sie sich im Clipboard-Fenster befinden und Alt F10 oder die rechte Maustaste drücken, erhalten Sie das nachfolgend beschriebene Menü. Alternativ können Sie Strg und die hervorgehoben dargestellte Taste eines Befehls aus dem lokalen Menü drücken, um einen Befehl direkt auszuführen.

Tabelle 2.3 Die Befehle des lokalen Menüs im Clipboard-Fenster

Befehl	Beschreibung
Inspect	Positioniert den Cursor an die Stelle des Fensters, aus dem der Eintrag kopiert wurde.
Remove	Entfernt den hervorgehobenen Eintrag. Entf hat dieselbe Wirkung.
Delete All	Löscht alle Einträge aus der Zwischenablage.

Befehl	*Beschreibung*
Freeze	Verhindert die automatische Anpassung der Einträge in der Zwischenablage an die Programmwerte.

Dynamische Anpassung In der Zwischenablage werden alle Einträge automatisch an die aktuellen Werte im Programm angepaßt, zum Beispiel ein Ausdruck aus einem *Watches* -Fenster, eine Variable aus einem *Inspector*-Fenster, oder ein Register aus dem CPU-Fenster. Sie können sich die Zwischenablage auch als eine Art vergrößertes *Watches* -Fenster vorstellen, in dem Sie den Wert einzelner Einträge gegebenenfalls festhalten können.

Sie könnten zum Beispiel einen Ausdruck aus dem *Watches* -Fenster in die Zwischenablage übernehmen. Dazu nehmen Sie den Ausdruck zunächst in das *Watches* -Fenster auf und drücken dann Umschalt F3 , um den Ausdruck in die Zwischenablage zu kopieren. Der Wert dieses Eintrags ändert sich dann in gleicher Weise, wie Sie dies vom *Watches* -Fenster kennen, falls Sie den Wert nicht mit dem Befehl *Freeze* des lokalen Menüs festschreiben. (Weitere Informationen erahlten Sie in Kapitel 6, »Programmdaten untersuchen und ändern«.)

Ein Vorteil der Überwachung von Ausdrücken in der Zwischenablage liegt darin, daß Sie den Ausdruck bei einem bestimmten Wert festhalten können, das Programm weiterlaufen lassen und diesen Wert dann später mit den veränderten Werten des *Watches* -Fensters vergleichen können.

Hinweise zur Verwendung der Zwischenablage Die Möglichkeiten der Zwischenablage sind sehr vielfältig, so daß sie hier nicht alle aufgeführt werden können. Nachfolgend einige Beispiele:

- Das Wiederauffinden bestimmter Zeilen in *Module*-Fenstern, indem Sie einen Eintrag aus der Zwischenablage in das Eingabefeld des Dialogfensters kopieren, das durch den Befehl *Goto* geöffnet wird.

- Überwachen eines Ausdrucks.

- Ändern von Variablenwerten im Dialogfenster *Data/Evaluate* oder beim Befehl *Change* des *Inspector*- oder *Watches* -Fensters.

- Einfügen von Strings in das *Log*-Fenster, um festzuhalten, was während einer Debugger-Sitzung geschah.

- Einfügen einer Adresse (der Position eines Objekts) in eine der Stellen, an denen Adressen abgefragt werden (wie im Feld *Address* des Dialogfensters *Breakpoints/Options* oder dem Dialogfenster *Run/Execute To*).

- Einfügen von Ausdrücken für die Bedingungen und Aktionen von Breakpoints.

- Einfügen von Parametern in das Dialogfenster *Run/Arguments*.

- Einfügen einer Windows-Prozedurbezeichnung oder eines OWL-Objektnamens in das Fenster *Windows Messages* .

- Einfügen eines Strings in das Dialogfenster beim *Search*-Befehl des *Module*-Fensters.

- Kopieren und Einfügen von Daten aus dem Datenausschnitt des CPU-Fensters.

- Kopieren von Code aus einem Teil des CPU-Fensters in einen anderen und Starten des Programm mit dem kopierten Code.

Das Hilfesystem

Wie Sie sehen, erleichtert TDW die Fehlersuche enorm. Man braucht sich nur sehr wenige Befehle zu merken. TDW speichert die vorangegangenen Eingaben, so daß sie später wiederverwendet werden können, und Sie können Makros definieren. Zudem arbeitet man mit einem ausgefeilten Fenstersystem. TDW ist ein sehr umfangreiches Programm und stellt zahlreiche Möglichkeiten und Befehle zur Verfügung. Um Unklarheiten zu vermeiden, bietet TDW die folgenden Hilfsfunktionen an:

- Eine Statusanzeige in der rechten oberen Ecke zeigt immer den aktuellen Vorgang an. Wenn sich zum Beispiel der Cursor in einem Fenster befindet, dann steht in der Vorgangsanzeige `READY` (engl. bereit). Ist ein Menü sichtbar, erscheint `MENU`. Wenn ein Dialogfenster aktiviert ist, steht dort `PROMPT`. Wenn Sie einmal nicht wissen, was in TDW geschieht, brauchen Sie sich nur die Statusanzeige anschauen. (Weitere Möglichkeiten für die Statusanzeige sind `SIZE/MOVE`, `MOVE`, `ERROR`, `RECORDING`, `WAIT`, `RUNNING`, `MENU`, `HELP`, `STATUS` und `PLAYBACK`.)

- Merken Sie sich, daß das aktive Fenster immer das oberste Fenster ist und von einer doppelten Linie eingerahmt wird.

- Durch das Drücken von F1 können Sie auf ein unfangreiches, kontextbezogenes Hilfesystem zugreifen. Wenn Sie anschließend noch einmal F1 drücken, erscheint ein Index der verfügbaren Hilfethemen, aus denen Sie das gewünschte Thema auswählen können.

- Mitkönnen Sie auf ein umfangreiches, kontextbezogenes Hilfesystem zugreifen.

- Die unterste Zeile des Bildschirms liefert Ihnen immer einen schnellen Überblick über die möglichen Tastaturbefehle. Die Zeile hängt ebenfalls vom jeweiligen Kontext ab. Sie ändert sich auch, wenn Sie die Tasten Alt oder Strg drücken.

Zu den beiden letzten Punkten erhalten Sie in den nachfolgenden Abschnitten noch weitere Informationen.

Online-Hilfe

TDW enthält, wie alle anderen Borland-Produkte, ein kontextbezogenes Hilfesystem, das mit einem Tastendruck jederzeit online verfügbar ist. Sie

erhalten also zu jedem beliebigen Zeitpunkt Hilfestellungen. Ob Sie sich in einem Menü, einem Fenster oder in einer Dialogfenster befinden oder ob Sie eine Fehlermeldung erhalten, spielt dabei keine Rolle.

Mit F1 erscheint ein Hilfefenster, das Informationen anzeigt, die dem aktuellen Kontext entsprechen. Mit einer Maus kann man auch einfach F1 in der Statuszeile anzeigen. Hilfefenster enthalten hervorgehobene Stichwörter, mit denen man weitere Informationen abrufen kann. Verwenden Sie die Cursortasten, um zum gewünschten Stichwort zu gelangen, und drücken Sie die Eingabetaste CR, um die Informationen zu diesem Stichwort abzurufen. Mit Hilfe der Tasten Pos1 und Ende können Sie zum ersten beziehungsweise letzten Stichwort auf dem Bildschirm springen.

Hilfestellungen erhalten Sie auch durch die Auswahl des Menüs *Help* in der Menüleiste (Alt H).

Alt F1 dient zur Rückkehr in den vorherigen Hilfsbildschirm des Hilfesystems. Innerhalb des Hilfesystems kann man mit Bild↑ die letzten 20 Hilfsbildschirme zurückgehen. Bild↓ funktioniert nur in einer Gruppe von verwandten Bildschirmen. Um den Index des Hilfesystems zu erhalten, müssen Sie Umschalt F1 (innerhalb des Hilfesystems F1) betätigen. Um sich über die Informationen des Hilfesystems zu informieren, wählen Sie *Help/Help on Help* aus dem Hauptmenü. Mit Esc verlassen Sie das Hilfesystem.

Die Statuszeile

Wo auch immer Sie sich in TDW befinden, in der untersten Zeile des Bildschirms erscheint grundsätzlich immer eine kurze Übersicht über die möglichen Tastaturbefehle. Damit haben Sie, dem Kontext entsprechend, immer die wichtigsten Tastaturbefehle auf einen Blick.

```
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

Im Fenster Die Statuszeile zeigt alle Befehle an, die mit den Funktionstasten ausgeführt werden können:

```
Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-Instr F8-Rtn F9-To F10-Local
```

Wenn Sie die Alt -Taste eine oder zwei Sekunden lang gedrückt halten, werden die Befehle angezeigt, die mit der Kombination Alt -Funktionstaste ausgeführt werden können.

```
Ctrl: I-Inspect W-Watch M-Module F-File P-Previous L-Line S-Search N-Next
```

Wenn Sie die Strg -Taste eine oder zwei Sekunden lang gedrückt halten, werden die Befehle angezeigt, die mit der Kombination Strg -Funktionstaste ausgeführt werden können.

Diese Hilfsmenüzeile ändert sich entsprechend dem aktuellen Fenster und dem aktuellen Fensterausschnitt und zeigt die Kurzformen der Befehle der lokalen Menüs an. Gibt es in einem lokalen Menü mehr Befehle als Erklärungen

in einer Zeile Platz haben, werden nur die ersten Befehle dargestellt. Mit Alt F10 können Sie alle verfügbaren Befehle eines lokalen Menüs durch Anzeige des gesamten Menüs aktivieren.

Wenn Sie eine Maus haben, brauchen Sie nur in die Befehle der Statuszeile zu klicken, um einen Alt - oder Strg -Befehl auszuführen.

In Menüs oder Dialogfenstern Jedesmal, wenn Sie sich in einem Menü oder Dialogfenster befinden, enthält die Statuszeile eine einzeilige Erklärung des aktuellen Menüpunkts. Wenn zum Beispiel *View/Registers* aktiviert ist, steht in der Statuszeile die Meldung `Open a CPU registers window`.

In der Statuszeile erhalten Sie auch für lokale Menübefehle Hilfestellungen.

3. Einführung in das Debuggen

Falls Sie zu den Menschen zählen, die TDW möglichst schnell ausprobieren möchten, dann ist dieses Kapitel genau das richtige für Sie. Hier wird Ihnen genügend Wissen darüber vermittelt, wie Sie ein Programm debuggen können, auch wenn Sie noch nicht das ganze Handbuch durchgearbeitet haben. Wenn Sie die hier beschriebenen grundlegenden Konzepte erlernt haben, können Sie das restliche Know-How über die hochintegrierte, intuitive Benutzeroberfläche und das kontextbezogene Hilfesystem durch Ausprobieren erlernen.

In diesem Kapitel werden alle grundlegenden Eigenschaften von TDW erklärt. Nach der Beschreibung des Beipielprogramms TDDEMO, das sich auf den ausgelieferten Disketten befindet, erfahren Sie, wie man

- Programme startet und anhält;
- die Inhalte von Programmvariablen ermitteln kann;
- komplexe Datenobjekte, wie Arrays und Strukturen überprüft;
- den Inhalt von Variablen verändert.

Das Beispielprogramm

Anhand eines Beispielprogramms (TDDEMO) werden die zwei wichtigsten Vorgänge gezeigt, die man kennen muß, um ein Programm zu debuggen: wie man ein Programm unterbricht und wieder startet und wie man Variablen und Datenstrukturen durchsuchen kann. Das Programm selbst ist als Beispiel gedacht. Ein Teil des Codes und der Datenstrukturen wurde nur zur Demonstration der Fähigkeiten von TDW eingefügt. So wird zum Beispiel die Verwendung von Kommandozeilenparametern demonstriert, auch wenn diese hier nicht weiter benötigt werden.

Bei der Programmierung von TDDEMO wurde das *EasyWin* -Modul eingebunden, um Programmausgaben in einem Anwendungsfenster unter Windows durchführen zu können. TDDEMO ist kein Programm, das die Fähigkeiten eines echten Windows-Programms enthält, es veranschaulicht jedoch einige nützliche Konzepte von Turbo Debugger für Windows.

Das Beispielprogramm erwartet die Eingabe einiger Zeilen Text. Danach werden die eingegebenen Wörter und Zeichen gezählt. Am Ende der Programme erscheint schließlich eine kurze Statistik des eingegebenen Texts, einschließlich der durchschnittlichen Anzahl der Wörter pro Zeile und der Häufigkeit der einzelnen Buchstaben.

Vergewissern Sie sich, daß das Verzeichnis, in dem Sie arbeiten, beide Dateien enthält, die Sie für diesen Lehrgang brauchen. Für dieses Beispiel brauchen Sie die beiden Dateien TDDEMO.C und TDDEMO.EXE.

Den Debugger starten Zum Starten des Demo-Programms gehen Sie wie folgt vor:

1. Vergewissern Sie sich zunächst, daß Windows im Standardmodus oder im erweiterten 386er-Modus arbeitet, denn TDW kann im im Real-Modus nicht betrieben werden.

2. Öffnen Sie im Programm-Manager von Windows die Programmgruppe, in der TDW enthalten ist, und aktivieren Sie TDW durch zweimaliges Anklicken des Programmsymbols.

3. Wenn TDW geladen ist, wählen Sie *File / Open* aus und geben den vollständigen Pfad von TDDEMO ein (Zuvor müssen Sie wahrscheinlich noch die Datei TDDEMO.C mit den Debugger-Informationen compilieren.). Drücken Sie anschließend die Eingabetaste CR

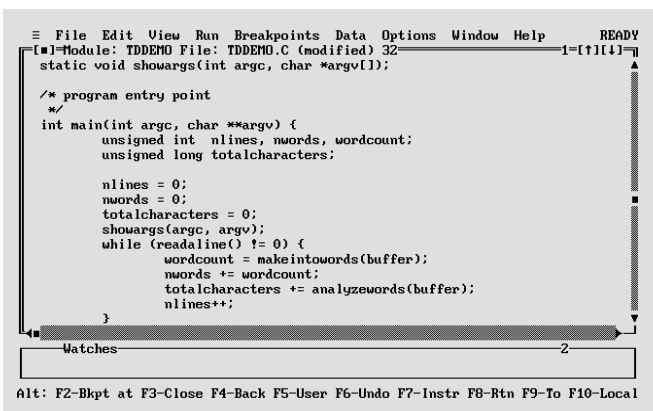


Abbildung 3.1 Der Bildschirm nach dem Start von TDDEMO

TDW lädt das Demoprogramm, zeigt den Startup-Bildschirm und setzt den Cursor an den Anfang des Programms. Der Startup-Bildschirm enthält die Menüleiste, das *Module*-Fenster, das *Watches* -Fenster und die Statuszeile.

Debugger verlassen Wenn Sie TDW verlassen wollen, dann geben Sie Alt X ein. Falls Sie sich während des Lehrgangs einmal hoffnungslos verirren sollten, können Sie mit Strg F2 das Programm neu laden und noch einmal beginnen. Strg F2 löscht jedoch weder Breakpoints noch Angaben im *Watches* -Fenster. Dafür müßten Sie Alt B D eingeben.

Hilfestellungen Mit der Taste F1 erhalten Sie jederzeit Hilfeinformationen, sei es über ein Fenster, einen Befehl, ein Dialogfenster oder eine Fehlermeldung. Man lernt eine Menge, wenn man sich Schritt für Schritt durch das Menüsystem bewegt und zu jedem Befehl mit F1 die entsprechende Erklärung abruft.

TDW im Einsatz

Die Menüleiste

In der obersten Zeile des Bildschirms befindet sich die Menüleiste. Um ein Menü daraus zu aktivieren, drücken Sie entweder die Taste F10, verwenden die Tasten ← und → zur Auswahl des Menüs und drücken CR, oder drücken die Taste Alt in Kombination mit dem ersten Buchstaben einer Menübezeichnung.

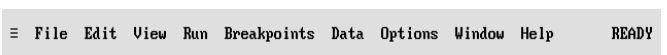


Abbildung 3.2
Die Menüleiste

Drücken Sie jetzt die Taste F10. Sie werden beobachten, daß der Cursor aus dem *Module*-Fenster verschwindet und daß der Befehl *File* in der Menüleiste hervorgehoben wird. Auch die unterste Bildschirmzeile verändert sich, um anzuzeigen, welche Befehle im *File*-Menü zur Verfügung stehen.

Benutzen Sie die Pfeiltasten, um im Menüsystem herumzuwandern. Geben Sie ↓ ein, um das in der Menüleiste hervorgehobene Menü zu aktivieren.

Ein Menü kann man auch durch Anklicken in der Menüleiste mit der Maus öffnen.

Drücken Sie jeweils Esc, um von den verschiedenen Ebenen des Menüsystems zurückzukehren. Ist nur eine Menübezeichnung auf der Menüleiste hervorgehoben, können Sie über Esc ins *Module*-Fenster zurückkehren. Die Menüleiste ist dann nicht mehr aktiviert.

Die Statuszeile

Die unterste Zeile des Bildschirms zeigt wichtige Funktionstasten und deren Auswirkung an.

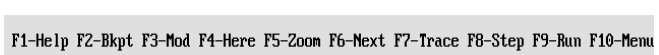


Abbildung 3.3
Die Statuszeile

Die Zeile verändert sich, je nachdem, welche Eingaben (Menübefehle, Eingaben in Dialogfenstern usw.) Sie machen. Halten Sie zum Beispiel die Alt-Taste

ein bis zwei Sekunden gedrückt. Sie sehen jetzt, daß sich die unterste Zeile verändert und die Funktionstasten zeigt, die mit Alt kombiniert werden können.

Drücken Sie nun eine Sekunde lang auf die Taste Strg. In der untersten Zeile erscheint eine Reihe von Befehlen. Es sind dies die Tastenkürzel der *Befehle des lokalen Men201s* im soeben aktivierten Fensterausschnitt. Diese Befehle ändern sich je nach Art des aktiven Fensters und Fensterausschnitts. Sie werden später mehr darüber erfahren.

Sobald Sie in das Menüsystem übergehen, ändert sich die Statuszeile erneut, um anzuzeigen, was die gerade hervorgehobene Option macht. Mit F10 kommen Sie in die Menüleiste. Drücken Sie danach einmal →, um das Menü *File* auszuwählen. In der Statuszeile steht nun die Meldung »File oriented functions«.

Mit ↓ können Sie die Optionen des *File*-Menüs durchgehen und verfolgen, wie sich die Meldung ändert. Drücken Sie Esc oder klicken Sie mit der Maus in das *Module*-Fenster, um das Menüsystem zu verlassen.

Die Fenster

Der Fensterbereich nimmt den größten Teil des Bildschirms ein. Hier untersuchen Sie einzelne Teile Ihres Programms, indem Sie es in verschiedenen Fenstern darstellen.

Zu Beginn erscheinen zwei Fenster auf dem Bildschirm: ein *Module*-Fenster und das *Watches* -Fenster. Solange man keine weiteren Fenster öffnet oder diese beiden Fenster nicht verschiebt, füllen sie zusammen die ganze Bildfläche aus, ohne sich zu überlappen. Neue Fenster überlappen automatisch die alten, bis sie verschoben oder gelöscht werden.

Abbildung 3.4 Die Fenster Module und Watches ohne Überlappung

Sie werden feststellen, daß das *Module*-Fenster durch eine doppelte Linie begrenzt und mit einer hervorgehobenen Überschrift versehen ist. Daraus ist ersichtlich, daß eben dieses Fenster gerade aktiviert ist. Mit Hilfe der Cursorta-

sten (Pfeiltasten, Pos1, Ende, Bild↑, Bild↓) können Sie sich innerhalb des aktivierten Fensters bewegen. Mit der Funktionstaste F6 wechselt man in ein anderes Fenster. Tun Sie das jetzt. Das *Watches* -Fenster wird aktiviert und erscheint mit einer unterlegten Überschrift und einer doppelten Begrenzungslinie. Drücken Sie anschließend ein weiteres Mal F6, um in das *Module*-Fenster zurückzukehren.

Verwenden Sie den Menübefehl *View* aus der Menüleiste, um weitere Fenster zu öffnen. Gehen Sie mit Alt V in das Menü *View* und geben Sie dann S ein, um ein *Stack*-Fenster zu öffnen. Das Menü verschwindet, während das *Stack*-Fenster über dem *Module*-Fenster erscheint.

Um das aktivierte Fenster zu löschen, drücken Sie bitte Alt F3 . Tun Sie das jetzt. Das *Stack*-Fenster verschwindet.

Das zuletzt geschlossene Fenster wird von TDW automatisch gespeichert. Sie können es also bei Bedarf jederzeit wieder abrufen. Falls Sie ein Fenster aus Versehen löschen, geben Sie Alt W ein, um ins Menü *Window* zu gelangen. Wählen Sie daraus durch Eingabe von U den Befehl *Undo close* und das *Stack*-Fenster erscheint wieder. Sie können auch Alt F6 eingeben, um das zuletzt geschlossene Fenster wieder zu restaurieren.

Im Menü *Window* stehen die Befehle, mit denen man die Fenster, die sich auf dem Bildschirm befinden, verändern kann. Man kann sowohl die Fenster auf dem Bildschirm umher bewegen, als auch ihre Größe verändern. (Diese Veränderungen können Sie auch mit Hilfe von Strg F5 vornehmen.)

Wählen Sie *Window / Size/Move* aus und verwenden Sie die Pfeiltasten, um das aktive Fenster (das *Stack*-Fenster) auf dem Bildschirm neu anzuordnen. Anschließend halten Sie die Taste Umschalt gedrückt. Nun können Sie durch Betätigen der Pfeiltasten die Größe des Fensters verändern. Drücken Sie die Eingabetaste CR, wenn Sie mit Größe und Position Ihres Fensters zufrieden sind.

Zur Vorbereitung auf den nächsten Abschnitt muß das Stack-Fenster durch Eingabe von Alt F3 wieder entfernt werden.

Das C-Beispielprogramm

Der ausgefüllte Pfeil in der linken Spalte des *Module*-Fensters zeigt an, an welcher Stelle TDW Ihr Programm angehalten hat. Da Sie noch nicht mit Ihrem Programm gearbeitet haben, zeigt der Pfeil auf die erste Zeile der Hauptroutine Ihres Programms. Geben Sie F7 ein, um eine einzelne Quellzeile abzuarbeiten. Nun befinden sich Pfeil und Cursor in der nächsten ausführbaren Zeile.

Beachten Sie die Angabe, die in der Überschrift des *Module*-Fensters rechts steht. Sie bezeichnet die Zeile, in der der Cursor steht. Wenn Sie nun den Cursor mit den Pfeiltasten auf und ab bewegen, können Sie beobachten, wie die Numerierung der jeweiligen Zeile in der Überschrift wechselt.

Um den Cursor auf eine beliebige Zeile des *Module*-Fensters zu setzen, drücken Sie Strg G und geben Sie eine Zeilennummer ein. Bstätigen Sie Ihre Eingabe mit CR

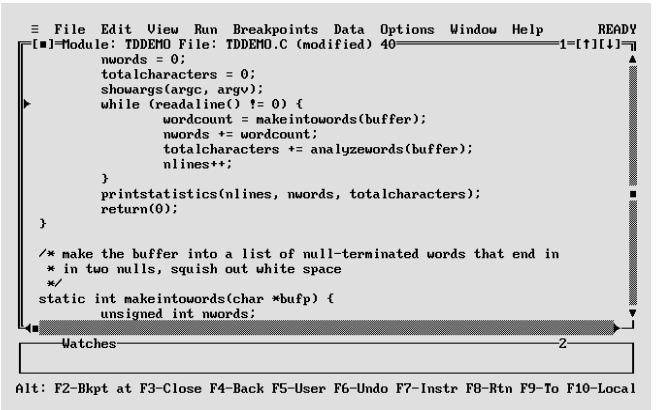


Abbildung 3.5 Das Programm hält sofort nach der Rückkehr von der Funktion argumente_ ausgeben an

Zur Kontrolle der Programmausführung stehen Ihnen zahlreiche Möglichkeiten (sehen Sie sich dazu bitte das *Run*-Menü an) zur Verfügung. Sie könnten beispielsweise den Wunsch haben, das Programm nur bis zur Zeile 39 ablaufen zu lassen. Positionieren Sie dazu zuerst den Cursor auf die Zeile 39 und drücken Sie dann F4. Das bewirkt das Ablaufen des Programms bis zu besagter Zeile 39, wobei die Anweisungen dieser Zeile selbst nicht ausgeführt werden. Betätigen Sie nun die Taste F7. Dadurch wird eine weitere einzelne Zeile Ihres Quellcodes abgearbeitet, in diesem Fall die Zeile 39, die die Funktion `showargs` aufruft. Der Cursor springt sofort zur Zeile 151, an der die Definition von `showargs` beginnt. Sie können nun die Funktion `showargs` durch fortwährendes Drücken von F7 vollständig abarbeiten, bis Sie zu der dem Funktionsaufruf folgenden Zeile 40 zurückgelangen. Stattdessen drücken Sie aber Alt F8, wodurch `showargs` ausgeführt wird und Sie dann zur Zeile 40 zurückgelangen, wo das Programm anhält. Dieser Befehl ist sehr nützlich, wenn Sie über das Ende einer Routine hinaus springen wollen.

Drücken Sie aber in Zeile 39 anstelle von F7 die Taste F8, so gelangen Sie ohne den Umweg über die Funktion `showargs` direkt zur Zeile 40. Die Taste F8 hat eine ähnliche Wirkung wie F7. Der Unterschied ist, daß Unterprogrammaufrufe zwar ausgeführt, der Quellcode jedoch nicht schrittweise abgearbeitet werden.

Um ein Programm bis zu einer bestimmten Stelle auszuführen, können Sie auch, anstatt eine Markierung durch den Cursor zu setzen, die Funktion oder die Zeilennummer direkt eingeben. Drücken Sie die Tasten Alt F9, um ein Programmsymbol zu bestimmen, zu dem Sie gehen wollen. Es erscheint ein Dialogfenster. Geben Sie `readaline` ein und drücken Sie die Eingabetaste. Das Programmm läuft dann bis zum Beginn der Funktion `readaline` (Zeile 142).

Breakpoints setzen

Die Verwendung von Breakpoints bietet Ihnen eine weitere Möglichkeit festzulegen, wo Ihr Programm anhält. Die einfachste Art, einen Breakpoint zu

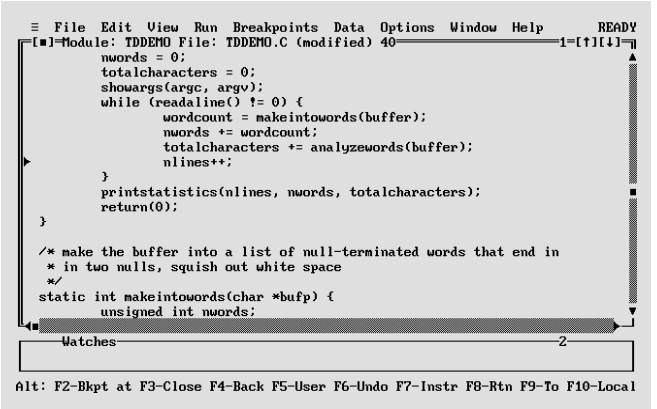


Abbildung 3.6
Ein Breakpoint in Zeile 44

setzen, ist die Eingabe von F2. Gehen Sie mit dem Cursor in Zeile 44 und betätigen Sie die Funktionstaste F2. TDW unterlegt nun die Zeile und zeigt damit an, daß hier ein Breakpoint gesetzt ist.

Sie können auch die Maus zum Ein-/Ausschalten von Breakpoints verwenden. Dazu klicken Sie in die ersten beiden Spalten des *Module*-Fensters.

Drücken Sie nun F9, um Ihr Programm ohne Unterbrechung auszuführen. Das Programm erscheint auf dem Bildschirm. Das DEMO-Programm läuft jetzt und wartet auf Ihre Texteingabe. Geben Sie `abc` ein, ein Leerzeichen, `def` und drücken Sie die Eingabetaste CR. Die Bildschirmdarstellung kehrt zu TDW zurück. Sie sehen nun den Pfeil neben der Zeile 44, an der Sie zum Abbrechen des Programms einen Breakpoint gesetzt haben.

Eine vollständige Beschreibung von Breakpoints, einschließlich der bedingten und globalen Breakpoints finden Sie in Kapitel 7, »Breakpoints«.

Das Watches-Fenster

Das *Watches* -Fenster am unteren Bildschirmrand zeigt den Wert von Variablen, die Sie zur Anzeige ausgesucht haben. Um beispielsweise den Wert der Variablen `nwords` festzustellen, bewegen Sie den Cursor auf die Variable in Zeile 42 und geben Sie Strg W ein. Dies ist die Abkürzung für den Befehl Alt F10 W im lokalen Menü.

Mit der Maus klicken Sie einfach Strg W in der Statuszeile an. Die Variable `nwords` erscheint jetzt im Fenster *Watches* mit der Angabe ihres Datentyps (`unsigned int`) und dem Inhalt. Während Sie Ihr Programm ausführen, bringt TDW diesen Wert immer wieder auf den neuesten Stand, um stets den aktuellen Wert der Variablen anzuzeigen.

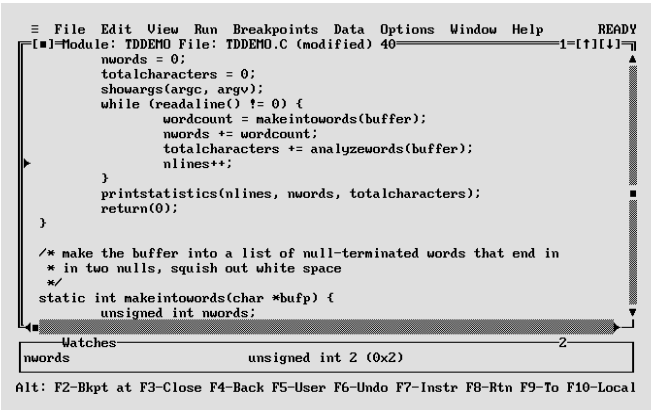


Abbildung 3.7
Eine C-Variable im Watches-Fenster

Einfache Datenobjekte in C untersuchen

Haben Sie Ihr Programm einmal auf diese Weise angehalten, bieten sich Ihnen mit Hilfe des Befehls *Inspect* eine Anzahl von Möglichkeiten, Daten zu untersuchen. Dieser besonders leistungsfähige Teil des Debugger erlaubt es Ihnen, Daten genauso zu überprüfen, wie Sie sie beim Programmieren vor Augen haben.

Mit den *Inspect*-Befehlen (in den verschiedenen lokalen Menüs und im Menü *Data*) können Sie jede Variable untersuchen. Angenommen, Sie wollen sich den Wert der Variablen nlines anschauen, so bewegen Sie den Cursor auf einen der Buchstaben von nlines und drücken Strg I . Nun erscheint ein *Inspector*-Fenster.

In der ersten Zeile steht der Name der Variablen, in der zweiten Zeile die Segment- und Offsetadresse. Die dritte Zeile zeigt den Datentyp von `nlines` (in diesem Fall der Datentyp `unsigned int`). Rechts davon können Sie den aktuellen Wert der Variablen ablesen.

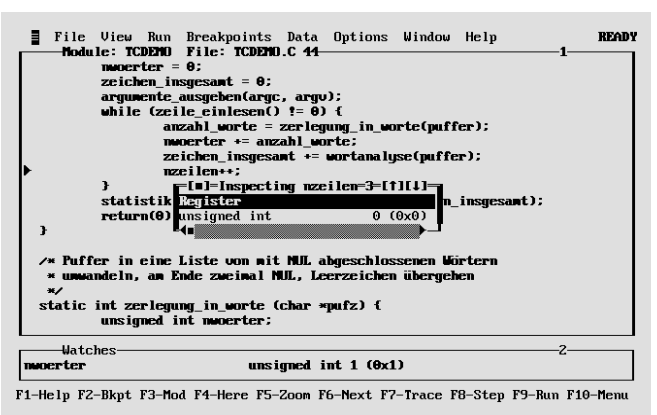


Abbildung 3.8
Ein Inspector-Fenster

Wenn Sie mit der Überprüfung der Variablen fertig sind, drücken Sie Esc, um das *Inspector*-Fenster zu schließen. Genau wie alle anderen Fenster können Sie das *Inspector*-Fenster auch mit Alt F3 schließen, oder das Schließfeld mit Ihrer Maus anklicken.

Lassen Sie uns diesen Vorgang kurz zusammenfassen. Durch die Eingabe der Abkürzung Strg gelangten Sie zu den Befehlen des lokalen Menüs im *Module*-Fenster. Dann wählten Sie mit I den Befehl *Inspect*.

Um Daten zu inspizieren, die nicht gleich so günstig im *Module*-Fenster erscheinen, drücken Sie bitte Alt D I . Damit erhalten Sie ein Dialogfenster auf dem Bildschirm, in dem Sie die Variable eingeben, die Sie untersuchen möchten. Geben Sie nun `letterinfo` ein und drücken Sie die Eingabetaste CR Es erscheint ein *Inspector*-Fenster, das die Werte der Elemente des Array `letterinfo` zeigt. Die erste Zeile unter dem Titel zeigt die Adresse des ersten Array-Elements von `letterinfo` im Hauptspeicher (Beachten Sie, daß die Adresse in Ihrem System möglicherweise einen anderen Wert hat). Gehen Sie mit Hilfe der Pfeiltasten durch die 26 Elemente, aus denen `letterinfo` besteht. Die Überschrift des *Inspector*-Fensters zeigt den Namen der Daten, die Sie gerade untersuchen. Im nächsten Abschnitt wird erläutert, wie man solch eine zusammengesetzte Datenstruktur untersucht.

Zusammengesetzte Datenstrukturen in C untersuchen

Ein zusammengesetztes Datenobjekt, wie zum Beispiel ein Array oder eine Struktur, enthält verschiedenartige Komponenten. Gehen Sie auf das vierte Element des Arrays `letterinfo` (also auf das Element, das mit `[3]` gekennzeichnet ist). Geben Sie Alt F10 ein, um in das lokale Menü des *Inspector*-Fensters zu gelangen und drücken Sie dann I, um den Befehl *Inspect* auszuwählen. Ein neues *Inspector*-Fenster erscheint und zeigt die Inhalte dieses Array-Elements. Das *Inspector*-Fenster zeigt die Inhalte einer Struktur des Typs `linfo`.

Wenn Sie den Cursor auf einen der Feldnamen positionieren, erscheint der Datentyp dieses Feldes im unteren Ausschnitt *Inspector*-Fensters. Würde eines

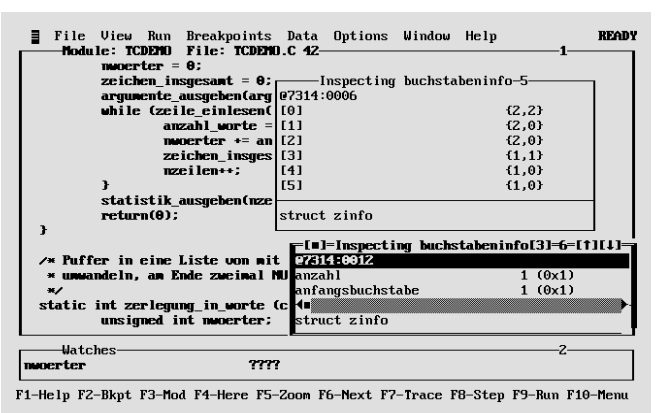


Abbildung 3.9 Die Untersuchung einer Struktur

dieser Fenster wiederum eine zusammengesetzte Datenstruktur enthalten, könnten Sie wieder einen *Inspect*-Befehl ausführen und noch tiefer in die Datenstruktur einblicken.

Drücken Sie Alt F3 , um beide *Inspector*-Fenster zu schließen und in das *Module*-Fenster zurückzukehren. (Durch die Eingabe von Alt F3 können Sie bequem mehrere *Inspector*-Fenster auf einmal entfernen, während Esc immer nur das zuletzt bearbeitete Fenster schließt.)

C-Daten ändern

Bisher haben Sie gelernt, wie Sie Daten des Programms untersuchen können. Nun geht's ans Ändern der Daten.

Gehen Sie mit den Pfeiltasten in der Quelldatei auf Zeile 38. Plazieren Sie den Cursor auf die Variable `totalcharacters` und drücken Sie Strg I , um den Wert dieser Variablen zu betrachten. Bei geöffnetem *Inspector*-Fenster drücken Sie nun Alt F10 , um in das lokale Menü des *Inspector*-Fensters zu gelangen. Geben Sie C ein, um die Option *Change* auszuwählen (über Strg C wäre dies auch direkt möglich gewesen). Nun erscheint ein Dialogfenster mit einem Eingabefeld für den neuen Wert.

An dieser Stelle können Sie jeden C-Ausdruck verwenden, der eine Zahl ergibt. Schreiben Sie `totalcharacters+4` und drücken Sie die Eingabetaste CR Im *Inspector*-Fenster wird jetzt der neue Wert `10L (0xA)` angezeigt.

Um Daten zu verändern, die nicht im *Module*-Fenster stehen, drücken Sie Alt D E . Es erscheint ein Dialogfenster, in dem der Name der zu ändernden Variablen einzugeben ist. Schreiben Sie `nlines` und drücken Sie die Eingabetaste CR Drücken Sie dann zweimal die Tab -Taste, um zum Eingabefeld *New Value* zu gelangen. Schreiben Sie `123` hinein und drücken Sie die Eingabetaste CR noch einmal. Das ganzzahlige Ergebnis in der zweiten Zeile wird dadurch zu `int 123 (0x7B)`.

Das war eine kurze Einführung in die Anwendung von TDW mit einem Programm, das mit Hilfe von Turbo C++ für Windows geschrieben wurde. In

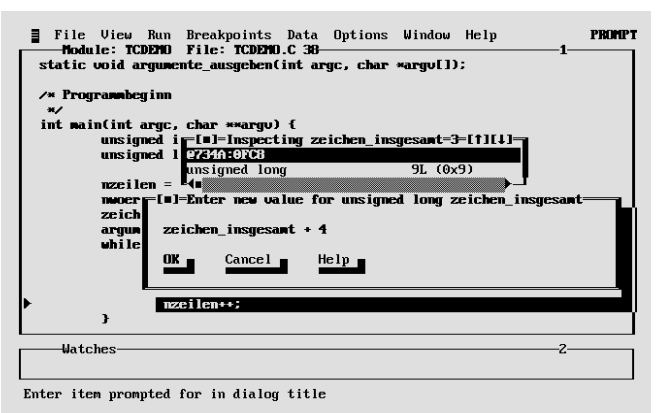


Abbildung 3.10 Das Dialogfenster Change zum Ändern von Daten

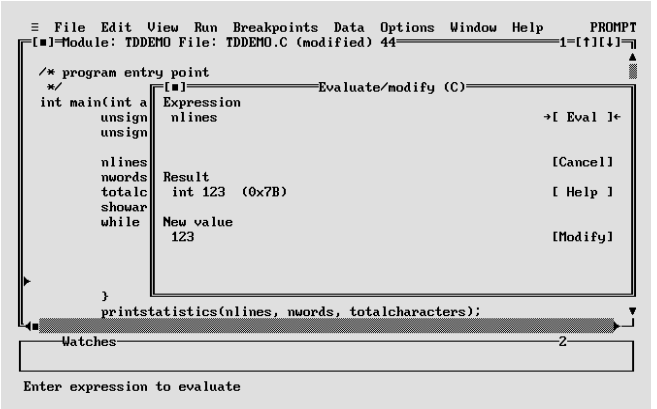


Abbildung 3.11
Das Dialogfenster von Evaluate/Modify

Kapitel 14, »Ein Standard-C-Programm debuggen«, und in Kapitel 15 »Ein ObjectWindows-Programm debuggen« finden Sie ausführlichere Beispiele für eine Debugger-Sitzung.

4. Der Aufruf von von Turbo Debugger

In diesem Kapitel erfahren Sie, wie Programme für das Debuggen vorbereitet werden. Zunächst beschreiben wir, wie Turbo Ddebuger für Windows gestartet wird und wie man die zahlreichen Kommandozeilenoptionen des Debuggers entsprechend dem zu untersuchenden Programm einstellt. Weiterhin wird erklärt, wie man diese Einstellungen in einer Konfigurationsdatei speichern kann. Sie lernen schließlich, wie Sie nach getaner Arbeit wieder zu Windows zurückkehren.

Ein Programm zum Debuggen vorbereiten

Wenn Sie Ihre Programme mit Turbo C++ für Windows compilieren und linken, sollten Sie alle Debug-Informationen vom Compiler erzeugen lassen. Haben Sie die einzelnen Module Ihres Programms ohne diese Informationszusätze compiliert, dann müssen Sie alle Module neu compilieren, um für das ganze Programm die Möglichkeit des Debuggens auf Quelltextebene zu schaffen. Sie können die Erzeugung der Debug-Informationen auf spezielle Module beschränken, aber es kann später recht ärgerlich sein, wenn man beim Debuggen in ein Modul gerät, das keine Debug-Information enthält. Deshalb empfehlen wir Ihnen, alle Module erneut zu compilieren.

Turbo C++ Programme vorbereiten

Wenn Sie mit der integrierten Entwicklungsumgebung des Turbo C++ Compilers arbeiten, ist die Erzeugung aller Debug-Informationen gemäß Voreinstellung aktiviert. Sollten Sie diese Einstellung geändert haben, so müssen Sie folgende Änderungen vornehmen:

1. Wählen Sie *Options/Linker/Settings* und markieren Sie im daraufhin geöffneten Dialogfenster das Feld *Include Debug Information* .

2. Wählen Sie *Options/Compiler/Advanced Code Generation* und markieren Sie das Feld *Debug Info* . Sie können anstatt dessen auch die Pragma-Anweisung *options*

-v verwenden, um die Debug-Informationen in die gewünschten Modulen aufzunehmen:

```
#pragma option -v
```

Den Debugger starten

Es gibt vier verschiedene Möglichkeiten, um TDW zu starten:

- Wenn Turbo C++ für Windows aktiviert ist, können Sie das Programm im aktiven Fenster mit dem Befehl *Start/Debugger* starten.

- Wenn die Windows-Ebene aktiv ist, öffnen Sie ganz einfach die Programmgruppe von TDW und starten den Debugger mit einem Doppelklick auf das entsprechende Programmsymbol. Mit dem Befehl *File/Open* laden Sie anschließend das Programm, das Sie debuggen wollen.
  Achtung: Bei den nächsten beiden Möglichkeiten muß sich sowohl TDW als auch das Program, das untersucht werden soll, im aktuellen Verzeichnis befinden.

- Wenn die Windows-Ebene aktiv ist, und Sie Kommandozeilenoptionen für TDW angeben wollen, können Sie TDW im Programm-Manager mit dem Befehl *Datei/Ausf201hren* starten, indem Sie das entsprechende Dialogfenster öffnen und den Befehl für TDW sowie die gewünschten Kommandozeilenoptionen eingeben.

- Von der DOS-Ebene aus können Sie TDW durch folgende Eingabe starte:

```
WIN TDW [Optionen] [Programmname [Programmargumente]]
```

Kommandozeilenoptionen angeben

Wenn Sie TDW von DOS aus starten oder den Befehl *Datei/Ausf201hren* im Programm-Manager verwenden, können Sie in der Eingabezeile nach dem Programmbefehl TDW zusätzliche Optionen angeben.

Falls Sie TDW unter Turbo C++ für Windows starten, können Sie die erforderlichen Kommandozeilenoptionen im Befehl *Start/Debugger Argumente* angeben.

Die direkte Eingabe von Kommandozeilenoptionen:
Das allgemeine Format der Kommandozeile lautet:

```
TDW [Optionen] [Programmname [Programmargumente]]
```

Die Angaben in Klammern sind optional. Wenn Sie solche Angaben verwenden, schreiben Sie sie bitte ohne Klammern. *Programmname* ist der Name des

Programms, das untersucht werden soll. Hinter dem Programmnamen können zusätzlich einige Optionen für den Debugger folgen. Hier sehen Sie einige Beispiele für Kommandozeilenaufrufe:

Befehl	Aktion
tdw -tc:\prog1 prog1 a b	Startet den Debugger, legt `c:\prog1` als aktuelles Verzeichnis fest und lädt das Programm `prog1` mit den zwei Argumenten `a` und `b`.
tdw prog2 -x	Startet den Debugger mit den Standardoptionen und lädt das Programm `prog2` mit dem Argument `-x`.

Wenn Sie nur `TDW` eingeben, wird TDW nur mit den Standardoptionen geladen.

Kommandozeilenoptionen unter Turbo C++ für Windows

Wenn Sie TDW von Turbo C++ für Windows aus starten und Kommandozeilenoptionen angeben wollen, wählen Sie den Befehl *Start/Debugger Argumente* . Geben Sie dabei keinen Programmnamen und keine Programmargumente an.

Das Programm, das Sie in TDW starten wollen, muß sich im aktuellen Editor-Fenster befinden. Wenn Sie zusätzlichliche Argumente für dieses Programm angeben wollen, wählen Sie *Start/Argumente* und geben die gewünschten Argumente in Fenster *Programmargumente* ein.

Hinweise, die Sie beachten sollten

- Bitte beachten Sie, daß Sie bei der Arbeit mit TDW sowohl die ».EXE«-Datei als auch alle verfügbaren originalen Quelldateien brauchen. TDW sucht zuerst in dem Verzeichnis nach Quelldateien, in dem der Compiler sie beim Compilieren gefunden hat, zweitens in dem Verzeichnis, das unter *Options/Path for Source* angegeben wurde, drittens im aktuellen Verzeichnis und viertens in dem Verzeichnis, in dem sich die ».EXE«-Datei befindet.

- Der Quellcode muß bereits zusammen mit allen benötigten Debug-Informationen in eine Programmdatei (».EXE«) übersetzt worden sein, bevor Sie mit dem Debuggen in TDW beginnen können.

- TDW kann nur für Windows-Programme verwendet werden, die mit einem Borland-Compiler erstellt wurden.

- Wenn Sie Ihr Programm unter Windows aufrufen und einen Fehler feststellen, müssen Sie es zunächst einmal beenden. Erst danach können Sie zur Suche nach der Fehlerquelle TDW starten.

Turbo Debugger und Windows

Wenn Sie TDW starten, werden Sie feststellen, daß der Debugger nicht im Grafik- sondern im Textmodus arbeitet. Dabei nimmt der Debugger den gesamten Bildschirmbereich ein, beschränkt sich also nicht wie andere Windows-Programme auf einen Bildschirmausschnitt. Trotz dieser Erscheinungsweise ist TDW ein Windows-Programm und nur unter Windows ablauffähig.

Im Gegensatz zu anderen Windows-Anwendungen sind die Windows-Tastenkürzel wie Alt Esc oder Strg Esc nicht verfügbar, um von TDW aus in andere Programme umzuschalten. Wenn jedoch das Anwenderprogramm aktiv ist, das Sie untersuchen, können Sie mit Alt Esc , Strg Esc oder mit der Maus in andere Programme wechseln.

Hinweis: Wollen Sie mittels Strg Esc ein unter Windows aktives Programm verlassen, erscheint dessen Name in der Task-Liste. In dieser Liste erscheint aber niemals der Name des Debuggers, denn TDW ist keine normale Windows-Task, in die Sie über die Task-Liste wechseln können.

Kommandozeilenoptionen

Alle Kommandozeilenoptionen beginnen mit einem Minuszeichen (-) und sind vom Programmbefehl `TDW` wie auch untereinander durch mindestens ein Leerzeichen getrennt. Wenn Sie eine Kommandozeilenoption abschalten wollen, können Sie das durch die Eingabe eines weiteren, der betreffenden Option folgenden Minuszeichens. Zum Beispiel schaltet *-vg-* die Zwischenspeicherung eines Grafikbildschirms aus. Das ist dann sinnvoll, wenn eine Option in der Konfigurationsdatei aktiviert wurde. Die Konfigurationsdatei können Sie modifizieren, indem Sie das Konfigurationsprogramm TDWINST verwenden, das in Anhang C, »Die Konfiguration von TDW«, beschrieben ist.

Im folgenden werden alle verfügbaren Kommandozeilenoptionen beschrieben. Anhang A, »Kommandozeilenoptionen«, enthält eine übersichtliche Aufstellung dieser Optionen.

Konfigurationsdatei laden (-c)

Durch diese Option wird die angegebene Konfigurationsdatei geladen. Zwischen *-c* und dem Dateinamen darf kein Leerzeichen stehen.

Wenn die Option *-c* nicht angegeben wird, lädt TDW die Datei TDCONFIG.TDW, falls diese vorhanden ist. Zum Beispiel:

```
TDW -cMEINECFG.TDW TDDEMO
```

Damit wird die Datei MEINECFG.TDW zusammen mit dem Quellcode des Programms TDDEMO geladen.

Bildschirm-umschaltung (-d)

Alle Optionen, die durch *-d* eingeleitet werden, beeinflussen die Art und Weise, wie die Umschaltung des Bildschirms durchgeführt wird.

-do Diese Option läßt den Debugger auf Ihrem zweiten Bildschirm laufen. Betrachten Sie den Bildschirm Ihres Programms auf dem ersten Monitor und lassen Sie den Debugger auf dem zweiten laufen.

-ds Die Kommandozeilenoption *-ds* ist die Standardoption (genannt *screen swapping*) für alle Bildschirme. Sie ist zwingend für Monochrombildschirme. Mit dieser Option wird eine separate Bildschirmoberfläche für den Debugger reserviert und eine andere für das Programm, das zu debuggen ist. Der ganze Bildschirm wird jedesmal aus dem Speicher geladen, wenn zwischen Programm und Debugger umgeschaltet wird. Dies ist zwar die zeitaufwendigste Methode für die Präsentation der beiden Bildschirmoberfächen, dafür ist sie sowohl auf allen Bildschirmen anwendbar als auch mit Programmen, die auf ungewöhnliche Art mit dem Bildschirm arbeiten.

Hilfestellungen (-h und -?)

Mit diesen Optionen erhalten Sie eine Bildschirmseite mit Hilfsinformationen, in denen die Kommandozeilenoptionen und die Kommandozeilensyntax von TDW beschrieben werden.

Assembler-Modus (-l)

Mit dieser Option wird TDW im Assembler-Modus gestartet. Der Startbildschirm enthält das *CPU*-Fenster. TDW führt den Startup-Code Ihres Programms nicht aus, der normalerweise sofort beim Laden abgearbeitet wird. Das bedeutet, daß Sie auch den Initialisierungscode untersuchen können.

Wenn Sie eine DLL debuggen und TDW mit der Option *-l* starten, können Sie auch den Assembler Startup-Code aller dynamischen Link-Bibliotheken (DLLs), die Ihr Programm aufruft, debuggen. (Mehr Informationen finden Sie in Kapitel 11, »Debuggen unter Windows«)

Maus-unterstützung (-p)

Die Option *-p* schaltet die Mausunterstützung ein. Da TDW standardmäßig auf Mausunterstützung eingestellt ist, brauchen Sie diese Option normalerweise nur dann anzugeben, wenn Sie die Maus abschalten wollen *(-p-)*.

Falls der Maustreiber von Windows nicht aktiviert ist, ist er für TDW ebenfalls nicht aktiviert und die Kommandozeilenoption *-p* wird nicht berücksichtigt.

Quellcode-Optionen (-s)

Alle Optionen, die mit *-s* beginnen, beeinflussen die Art und Weise, in der TDW Quelldateien und Programmsymbole verarbeitet.

-sc Die Option *-sc* ignoriert Groß- und Kleinschreibung bei der Eingabe von Symbolnamen, auch wenn Ihr Programm unter Berücksichtigung von Groß- und Kleinschreibung gelinkt wurde.

Ohne die Option *-sc* ignoriert TDW Groß- und Kleinschreibung nur dann, wenn Sie Ihr Programm mit der Option *Case ignore* gelinkt haben.

-sd Mit dieser Option können Verzeichnisse angegeben werden, in denen die Quelldateien liegen. Die Syntax lautet:

```
-sdVerzeichnis[;Verzeichnis...]
```

Um verschiedene Verzeichnisse zu wählen, geben Sie die einzelnen Verzeichnisse durch Strichpunkt voneinander getrennt hinter *-sd* an, oder Sie verwenden die Option *-sd* mehrmals. Die Verzeichnisse werden in der angegebenen Reihenfolge durchsucht. Der Suchpfad von *Verzeichnis* kann relativ oder absolut angegeben werden und auch eine Laufwerksangabe beinhalten. Wenn in der Konfigurationsdatei Verzeichnisse angegeben werden, so werden die Verzeichnisangaben, die mit Hilfe der Option *-sd* gewählt wurden, an diese Einträge angehängt. (Diese Option ändert das Startverzeichnis nicht.)

Start-verzeichnis (-t)

Diese Option ändert das Startverzeichnis von TDW, in dem TDW als erstes nach der Konfigurationsdatei und den ».EXE«-Dateien sucht, die nicht mit einem vollständigen Pfad angegeben wurden. Zwischen der Option und dem Pfad des Verzeichnisses darf kein Leerzeichen stehen.

Die Syntax lautet:

```
-tVerzeichnis
```

Sie können mit dieser Option nur ein Startverzeichnis vorgeben. Wenn Sie mehrere Verzeichnis in einem Eintrag von *-t* angeben, ignoriert TDW alle angegebenen Verzeichnisse. Wenn Sie diese Optionen mehr als einmal in derselben Kommandozeile angeben, verwendet TDW nur den letzten Eintrag.

Zum Beispiel verwendet TDW bei folgendem Aufruf das Verzeichnis *D:\ARBEIT*:

```
TDW  -tc:\utils\screen -td:\arbeit
```

Konfigurations-Dateien

TDW verwendet eine Konfigurationsdatei, die die Voreinstellungen für die Kommandozeilenoptionen enthält. Sie können TDINST zur Einstellung der Standardwerte für den Fall verwenden, daß keine Konfigurationsdatei vorhanden ist und eine Konfigurationsdatei erstellt werden soll. (In Anhang C, »Die Konfiguration von TDW«, finden Sie eine Beschreibung, wie man eine Konfigurationsdatei zu TDW für Windows erstellen kann.

TDW sucht die Konfigurationsdatei TDCONFIG.TDW zuerst im aktuellen Verzeichnis, dann in dem Verzeichnis, das in TDWINST angegeben wurde, und schließlich in dem Verzeichnis, aus dem TDW.EXE aufgerufen wurde.

Wenn TDW eine Konfigurationsdatei findet, werden die dort angegebenen Einstellungen verwendet. Jede Kommandozeilenoption, die beim Aufruf von TDW angegeben wird, hat jedoch den Vorrang gegenüber den Voreinstellungen.

Das Menü Options

Mit diesem Menü können Sie eine Anzahl von Parametern festlegen, die das allgemeine Erscheinungsbild und die Arbeitsweise von TDW kontrollieren. Die folgenden Abschnitte beschreiben alle Menübefehle und verweisen Sie, wenn nötig, auf andere Abschnitte des Handbuchs mit weiteren Informationen.

Das Auswahlfeld Language

In Kapitel 9, »Ausdrücke«, wird beschrieben, wie man die Programmiersprache für die Auswertung der Ausdrücke einstellt und wie dieser Befehl die Art und Weise beeinflußt, in der Ausdrücke eingegeben werden müssen.

Das Menü Macros

Dieser Befehl aktiviert ein weiteres Menü, mit dessen Hilfe Sie neue Tastaturmakros definieren können, oder er löscht Makros, die Sie bereits einer Taste zugeordnet hatten. Das Menü enthält die folgenden Befehle: *Create*, *Stop recording*, *Remove* und *Delete All*.

Create

Mit dem Befehl *Options/Macros/Create* (engl. erzeugen) schalten Sie die Aufzeichnung von Tastatureingaben ein, die einer Taste zugeordnet werden sollen. Alternativ können Sie auch das Tastenkürzel Alt = verwenden.

Wenn Sie *Create* aktivieren, werden Sie zunächst nach der Taste gefragt, der das Makro zugeordnet werden soll. Geben Sie an dieser Stelle den entsprechende Tastenkombination ein, z. B. (Umschalt F9). Solange die Aufzeichnungsfunktion aktiv ist, erscheint in der oberen rechten Ecke des Bildschirms die Meldung `Recording`. Dann geben Sie die Tastenfolge ein, die Sie aufnehmen wollen. Diese Eingaben werden von TDW genauso bearbeitet, als ob Sie kein Macro abspeichern würden.

Stop recording

Durch den Befehl *Stop recording* (engl. Aufnahme beenden) wird die Aufzeichnung von Tastenkombinationen, die einer Taste zugeordnet werden, beendet. Verwenden Sie dazu das Tastenkürzel Alt - oder die Tastenkombination, die Sie neu definieren wollen.

Achtung: Den Befehl *Options/Macro/Stop Recording* können Sie nicht verwenden, um Ihr Makro zu beenden, da diese Tastenfolge direkt in Ihr Makro eingefügt wird. (Der Menüpunkt dient lediglich als Erinnerungsstütze für das Tastenkürzel Alt -).

Remove

Mit dem Befehl *Remove* (engl. entfernen) können Sie ein Makro löschen, das einer Taste zugeordnet ist. Die Eingabeaufforderung fragt nach der Taste, deren zugeordnetes Makro Sie entfernen wollen.

Delete All

Der Befehl *Delete all* (engl. alle löschen) entfernt alle Makrodefinitionen für Tastenfolgen und stellt die urprünglichen Tastenbelegungen wieder her.

Das Dialogfenster Display options

Dieser Befehl öffnet ein Dialogfenster, in dem Sie verschiedene Optionen zur Bildschirmdarstellung von TDW einstellen können.

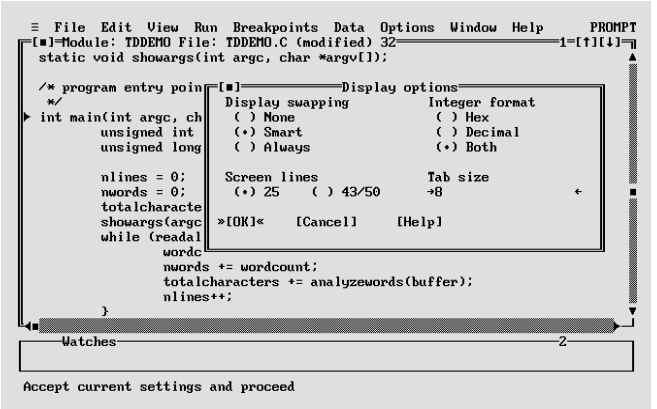


Abbildung 4.1 Das Dialogfenster Display Options

Display swapping

Mit den Auswahlfeldern unter *Display Swapping* können Sie zwischen zwei Arten der Bildschirmauslagerung wählen:

Smart Es wird nur dann auf den Benutzerbildschirm umgeschaltet, wenn Ihr Programm eventuell etwas auf den Bildschirm ausgibt. TDW wechselt die Bildschirme, sobald Sie eine Routine überspringen, oder wenn Sie einen Befehl oder eine Zeile ausführen, die unter Umständen aus dem Bildschirmspeicher liest oder dort hineinschreibt. Dies ist die Standardeinstellung.

Always Es wird immer auf den Benutzerbildschirm umgeschaltet, sobald das Benutzerprogramm läuft. Benützen Sie diese Option, wenn die Option *Smart* nicht bei allen Bildschirmausgaben Ihres Programms umschaltet. Unter dieser Option flackert Ihr Bildschirm jedesmal, wenn Sie eine Zeile in Ihrem Programm ausführen, da der Bildschirm von TDW dabei kurz durch Ihren Programmbildschirm ersetzt wird.

Integer format

Die Option *Integer format* ermöglicht es Ihnen, zwischen drei Formaten der Integer-Darstellungen zu wählen:

Decimal zeigt Integer-Zahlen als gewöhnliche Dezimalzahlen an.

Hex zeigt Integer-Zahlen als Hexadezimalzahlen in einem Format an, das der eingestellten Programmiersprache entspricht.

Both zeigt Integer-Zahlen sowohl als Dezimal- als auch als Hexadezimalzahlen an, die dann hinter dem dezimalen Wert in Klammern stehen.

Screen lines

Benützen Sie die Option *Screen lines* (engl. Bildschirmzeilen), um festzulegen, ob der Bildschirm von TDW mit dem normalen 25-ZeilenModus oder mit den

43- oder 50-Zeilen-Modi arbeitet, die auf EGA- und VGA-Bildschirmadaptern zur Verfügung stehen.

Tab Size
In diesem Eingabefeld können Sie festlegen, wieviele Spalten ein Tabulator umfassen soll. Sie können die Breite der Tabulatorspalten verringern, um mehr Text in Quelldateien darstellen zu können, die mit vielen Einrückungen versehen sind. Die mögliche Spaltenbreite liegt zwischen 1 und 32.

Der Befehl Path for source

Mit diesem Befehl werden die Verzeichnisse festgelegt, in denen TDW nach Ihren Quelldateien sucht. In Kapitel 8, »Dateien untersuchen und verändern«, finden Sie im Rahmen der Beschreibung des *Module*-Fensters weitere Informationen zu dieser Option.

Der Befehl Save options

Dieser Befehl speichert die Optionen, die Sie eingegeben haben, in einer Konfigurationsdatei auf dem Laufwerk. Dazu zählen:

- Ihre Makros;
- die aktuelle Fensteraufteilung;
- die Formate von Fensterausschnitten;
- alle Einstellungen, die im Menü *Options* vorgenommen worden sind.

TDW läßt Ihnen die Wahl zwischen drei Möglichkeiten, Ihre Einstellungen

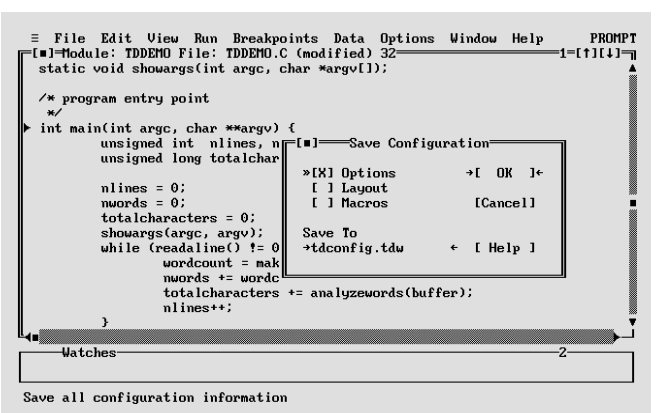


Abbildung 4.2 Das Dialogfenster Save Options

zu sichern, je nachdem, welche Markierungsfelder unter *Save Configuration* angekreuzt wurden:

Options	speichert alle Einstellungen des Menüs *Options*.
Layout	speichert nur die Anordnung der Fenster.
Macros	speichert nur die aktuell definierten Makros.

Sie können auch das Eingabefeld *Save to* verwenden, um einen anderen Namen für die Konfigurationsdatei anzugeben, in der die Optionen gespeichert werden sollen.

Der Befehl Restore options

Damit können Sie die zuvor von Ihnen abgespeicherten Einstellungen aus einer Datei lesen. Das können verschiedenartige Konfigurationsdateien sein, die verschiedene Makros, Fensteraufteilungen usw. enthalten. Sie müssen dabei eine Datei angeben, die mit dem Befehl *Save Options* oder von TDWINST erzeugt wurde.

Rückkehr zu Windows

Sie können jederzeit Ihre Debugger-Sitzung beenden und zum Windows-Programm-Manager zurückkehren, indem Sie Alt X eingeben. Sie erreichen dasselbe, wenn Sie den Debugger mit *File/Quit* verlassen. (In einem Dialogfenster müssen Sie vorher Esc drücken.)

5. Steuerung der Programmausführung

Will man ein Programm debuggen, geht man normalerweise so vor, das Programm stückweise auszuführen und dann zu überprüfen, ob es sich an einem bestimmten Punkt richtig verhält. TDW bietet Ihnen viele Möglichkeiten, die Ausführung Ihres Programms zu kontrollieren. Sie können

- einzelne Maschinensprachebefehle oder einzelne Quellzeilen ausführen;
- Unterprogrammaufrufe überspringen;
- den Debugger »animieren« (kontinuierlich Trace-Befehle ausführen);
- eine aktuelle Prozedur oder Funktion so lange ausführen, bis sie zum aufrufenden Programmteil zurückkehrt;
- eine Funktion oder Prozedur bis zu einem bestimmten Punkt ausführen;
- weitermachen, bis ein Breakpoint erreicht wird;
- das Programm rückwärts ausführen.

Eine Debugger-Sitzung besteht aus abwechselnden Phasen, in denen entweder Ihr Programm oder der Debugger läuft. Wenn der Debugger läuft, können Sie Ihr Programm starten, indem Sie eine Befehlsoption aus dem Menü *Run* wählen oder das entsprechende Tastenkürzel eingeben. Läuft dagegen Ihr Programm, kann der Debugger die Steuerung wieder übernehmen, wenn zum Beispiel ein spezieller Abschnitt Ihres Programms ausgeführt ist, oder wenn Sie die Ausführung mit einer speziellen Tastenfolge unterbrechen, oder schließlich, wenn TDW auf einen Breakpoint trifft.

Dieses Kapitel zeigt Ihnen, wie Sie den Zustand Ihres Programms untersuchen können, wenn der Debugger arbeitet. Sie lernen verschiedene Methoden kennen, um Teile Ihres Programms auszuführen bzw. während der Ausführung zu unterbrechen. Schließlich erfahren Sie, wie Sie eine neue Debugger-Sitzung starten, sowohl mit demselben als auch mit einem anderen Programm.

Den aktuellen Zustand des Programms untersuchen

Der »Zustand« Ihres Programms besteht aus folgenden Elementen:

- seinen Kommandozeilenargumenten;
- dem Stack mit den aktiven Funktionen und Prozeduren;
- der aktuellen Position im Quell- oder Maschinencode;
- den Registerwerten;
- dem Speicherinhalt;
- dem Grund, aus dem Debugger Ihr Programm gestoppt hat;
- den Werten der Variablen Ihres Programms.

Die folgenden Abschnitte erklären das *Variables* -Fenster, das *Stack*-Fenster, die lokalen Menüs der globalen und statischen Fensterausschnitte, den Befehl *Origin* und den Befehl *Get Info*. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen darüber, wie Sie Werte von Variablen in Ihrem Programm untersuchen und ändern können.

Das Fenster Variables

Sie öffnen das Fenster *Variables* , indem Sie über die Menüleiste den Befehl *View/Variables* auswählen. Dieses Fenster zeigt alle Variablen (Bezeichner und Werte), die Sie von der aktuellen Position in Ihrem Programmm aus erreichen können. Verwenden Sie es, um Variablen zu finden, an deren Schreibweise Sie sich nicht erinnern können. Dann können Sie ihre Werte mit Hilfe der Befehle aus dem lokalen Menü näher untersuchen oder ändern. Sie können dieses Fenster auch dazu verwenden, die lokalen Variablen zu allen Funktionen, die aufgerufen werden, zu untersuchen.

Abbildung 5.1
Das Variables- Fenster

Ein *Variables* -Fenster hat zwei Ausschnitte:

- einen globalen Ausschnitt (oben) mit allen globalen Symbolen Ihres Programms;
- einen statischen/lokalen Ausschnitt (unten) mit den statischen Symbolen des aktuellen Moduls (das Modul mit der aktuellen Programmposition, CS:IP) und den lokalen Symbolen der aktuellen Funktion.

In beiden Ausschnitten steht der Variablenbezeichner am linken Rand und der Wert am rechten Rand. Wenn TDW keine Informationen für das Symbol findet, erscheinen vier Fragezeichen (????).

Wie alle lokalen Menüs, können Sie auch das lokale Menü des globalen Ausschnitts durch die Eingabe von Alt F10 aufrufen. Falls die Tastenkürzel aktiviert sind, können Sie Strg in Kombination mit dem ersten Buchstaben des gewünschten Befehls drücken, um diesen auszuwählen.

Wenn Ihr Programm Funktionen mit rekursiven Aufrufen enthält oder wenn Sie die lokalen Variablen einer aufgerufenen Funktion anschauen möchten, können Sie die lokalen Daten einer Funktion untersuchen. Rufen Sie zunächst über *View/Stack* das *Stack*-Fenster auf, bewegen Sie dann den Cursor auf den gewünschten Punkt im Funktionsaufruf. Danach drücken Sie Alt F10 und wählen *Locals*. Der statische Ausschnitt aus dem *Variables*- Fenster zeigt nun die Werte für diesen speziellen Punkt der Funktion.

Das lokale Menü des globalen Fensterausschnitts

Dieses lokale Menü umfaßt drei Befehle: *Inspect*, *Change* und *Watch*.

Inspect Der Befehl *Inspect* (engl. untersuchen) öffnet ein *Inspector*Fenster, das den Inhalt des momentan unterlegten globalen Symbols zeigt. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen über die Arbeit mit *Inspector*-Fenstern.

Wenn die Variable, die Sie untersuchen wollen, der Name einer Routine ist, wird der Quellcode für diese Funktion gezeigt. Falls keine Quelldatei existiert, zeigt ein *CPU*-Fenster den disassemblierten Code.

Wenn sich der Name der untersuchten Variablen mit einer lokalen Variablen des gleichen Namens überschneidet, so sehen Sie nicht den Wert der lokalen Variablen, sondern den der globalen. Üblicherweise verhalten sich *Inspector*-Fenster etwas anders: Sie zeigen gewöhnlich den Wert einer Variablen innerhalb des aktuellen Bereichs im Programm (CS:IP). Durch das veränderte Verhalten haben jedoch Sie eine bequeme Möglichkeit, sich die Werte von globalen Variablen anzuschauen, deren Namen auch für lokale Variablen verwendet werden.

Change Der Befehl *Change* (engl. verändern) verändert den Wert des momentan ausgewählten Symbols in den Wert, der in einem Dialogfenster neu eingeben wird. TDW führt alle notwendigen Umwandlungen von Datentypen genauso aus, als würden Sie durch den jeweiligen Zuweisungsoperator der Sprache vollzogen, mit der Sie gerade arbeiten. In Kapitel 9, »Ausdrücke«, finden

Sie weitere Informationen über Zuweisungen und Veränderungen von Datentypen.

Sie können den Wert der gerade ausgewählten Variablen auch einfach dadurch ändern, daß Sie anfangen, einen neuen Wert einzugeben. Dadurch erscheint das gleiche Dialogfenster, das auch aktiviert wird, wenn Sie vorher den Befehl *Change* eingeben.

Watch Der Befehl *Watch* öffnet ein *Watches* -Fenster und stellt darin das momentan unterlegte globale Symbol dar. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie mehr Informationen zum *Watches* -Fenster.

Das *Watches* -Fenster verfolgt nicht, ob eine Variable global oder lokal ist. Wenn Sie eine globale Variable mit dem *Watch* -Befehl einfügen und später auf eine lokale Variable mit demselben Bezeichner treffen, zeigt das *Watches* -Fenster solange die lokale Variable, bis Sie deren Gültigkeitsbereich verlassen. Mit anderen Worten zeigt Ihnen das *Watches* -Fenster immer den Wert einer Variablen aus der Sicht der aktuellen Programmposition (CS:IP).

Das lokale Menü im statischen Fensterausschnitt

Geben Sie die Tastenfolge Alt F10 ein, um das lokale Menü des statischen Fensterausschnitts aufzurufen. Falls die Tastenkürzel aktiviert sind, gelangen Sie zu dem gewünschten Befehl, indem Sie die Strg -Taste zusammen mit dem ersten Buchstaben dieses Befehls drücken.

Der statische Fensterausschnitt verfügt über vier Menübefehle: *Inspect*, *Change*, *Watch* and *Show*.

Inspect Der Befehl *Inspect* (engl. untersuchen) öffnet ein *Inspector*Fenster, das den Inhalt des gerade ausgewählten lokalen Symbols zeigt. In Kapitel 6, »Programmdaten untersuchen und ändern«, finden Sie weitere Informationen über die Arbeit mit *Inspector*-Fenstern.

Change Der Befehl *Change* (engl. verändern) verändert den Wert des momentan ausgewählten lokalen Symbols in den Wert, der in einem Dialogfenster neu eingegeben wird. TDW führt alle notwendigen Unwandlungen von Datentypen genauso aus, als würden Sie durch den entsprechenden Zuweisungsoperator der Sprache vollzogen, mit der Sie gerade arbeiten. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über Zuweisungen und Veränderungen von Datentypen.

Sie können den Wert einer Variablen, die gerade ausgewählt ist, auch einfach dadurch ändern, daß Sie anfangen, einen neuen Wert einzugeben. Dadurch erscheint das gleiche Dialogfenster wie beim Befehl *Change*.

Watch Der Befehl *Watch* öffnet eine *Watches* -Fenster und stellt das momentan selektierte (unterlegte) statische oder lokale Symbol hinein. In Kapitel 6, »Programmdaten untersuchen und ändern« finden Sie mehr Informationen über das Arbeiten mit *Watches* -Fenstern.

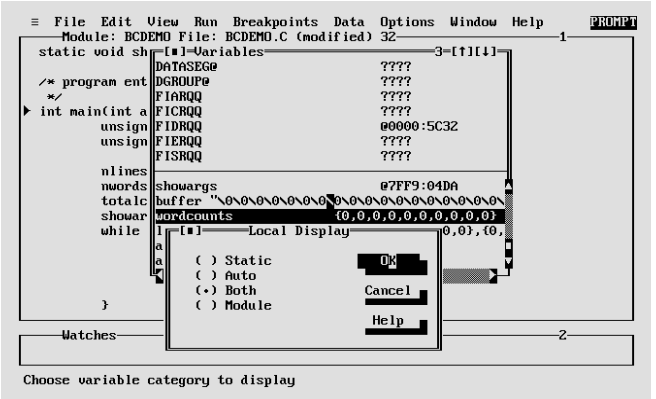


Abbildung 5.2 Das Dialogfenster Local Display

Show Der Befehl *Show* öffnet das Dialogfenster *Local Display* in dem Sie einstellen können, welche Variablen angezeigt werden (statische, automatische oder beide Arten) und aus welchem Modul diese Variablen kommen sollen.

Folgende Auswahlmöglichkeiten haben Sie in diesem Dialogfenster:

Static	Nur statische Variablen anzeigen.
Auto	Nur die lokalen Variablen des aktuellen Blocks anzeigen.
Both	Variablen beider Typen anzeigen (Standardeinstellung).
Module	Diese Auswahl öffnet ein Dialogfenster mit der Modulliste des Programms, aus der Sie ein neues Modul auswählen.

Das Fenster Stack

Ein *Stack*-Fenster kann mit dem Befehl *View / Stack* angelegt werden. Das *Stack*-Fenster enthält eine Liste mit allen aufgerufenen und aktiven Funktionen oder Prozeduren. Die zuletzt aufgerufene Routine wird in der Liste als erste dargestellt. Dahinter folgen die aufrufende Routine und dann deren aufrufende Prozedur, bis man zur der ersten Funktion oder Prozedur des Programms kommt (in Turbo C++ für Windows die Funktion `WinMain`). Zu jeder Funktion oder Prozedur sehen Sie die Werte der einzelnen Parameter, mit denen die Routine aufgerufen wurde.

Abbildung 5.3 Das Stack-Fenster

Im *Stack*-Fenster erscheinen die Namen der Methoden oder Elementfunktionen hinter der Angabe der entsprechenden Objekte oder Klassen, in denen die Methode oder Elementfunktion definiert ist:

```
Shapes::ACircle(174, 360, 75.0) /* C++ */
```

Wenn Sie das lokale Menü des *Stack*-Fensters aktivieren wollen, drücken Sie die Tastenkombination Alt F10 . Mit Strg und dem Anfangsbuchstaben des jeweiligen Befehls kann ein gewünschter Menübefehl auch direkt aufgerufen werden.

Das lokale Menü des Stack-Fensters
Dieses Menü stellt die beiden Befehle *Inspect* und *Locals* bereit.

Inspect Mit dem Befehl *Inspect* (engl. untersuchen) wird ein *Module*Fenster ab der aktivierten Zeile in der gerade ausgewählten Funktion geöffnet. Wenn die hervorgehoben dargestellte Funktion ganz oben steht (also zuletzt aufgerufen wurde), zeigt das *Module*-Fenster die aktuelle Programmposition (CS:IP). Gehört die ausgewählte Funktion zu einer der Funktionen, durch die die letzte Funktion aufgerufen haben, so wird der Cursor auf die Zeile im Fenster gesetzt, in der der nächste Befehl hinter dem Funktionsaufruf steht.

Dieser Befehl kann auch mit der Eingabetaste CR aufgerufen werden, wenn die gewünschte Funktion hervorgehoben dargestellt ist.

Locals Der Befehl *Locals* (engl. lokale [Variablen]) öffnet ein *Variables* -Fenster, in dem sowohl die lokalen Symbole des aktuellen Moduls als auch die Symbole der gerade ausgewählten Funktion dargestellt werden. Wenn sich eine Funktion rekursiv selbst aufruft, erscheint sie auch mehrmals im *Stack*-Fenster. Falls Sie den Auswahlbalken auf einen bestimmten Funktionsaufruf setzen, können Sie die lokalen Variablen dieses Funktionsaufrufs untersuchen.

Der Befehl Origin im lokalen Menü

Das *Module*-Fenster und der Code-Ausschnitt eines *CPU*-Fensters stellen beide den lokalen Befehl *Origin* (engl. Ursprung) in ihren lokalen Menüs bereit. Der Befehl *Origin* setzt den Cursor an die aktuelle Position des Programms (CS:IP). Das ist nützlich, wenn Sie im Code umhergewandert sind und wieder zu dem Punkt zurückwollen, an dem das Programm angehalten wurde.

Der Befehl Get Info

Mit dem Befehl *File/Get info* (engl. Informationen abrufen) erhalten Sie einen Überblick über die Speicherplatzbelegung und die Steuerung des Programms durch den Debugger. Diese und andere Informationen erscheinen in einem

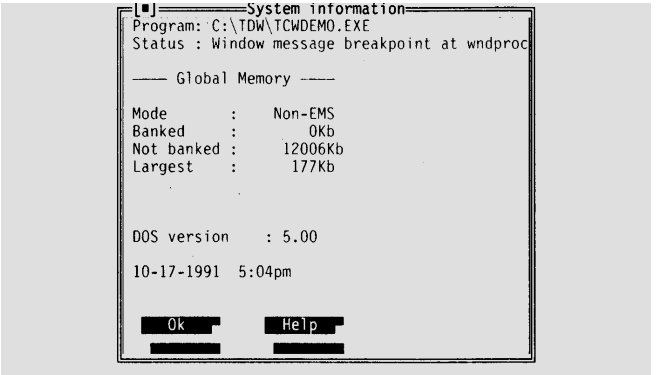


Abbildung 5.4 Die Informationen im Windows-Fenster Get Info

Informationsblock, der wieder entfernt wird, wenn Sie CR, Leertaste oder Esc drücken.

Der Informationsblock enthält:

- den Namen des Programms, das Sie debuggen;
- eine Beschreibung, wo und warum das Programm angehalten wurde;
- Informationen über Art und Umfang des auf Ihrem System verfügbaren Speicher;
- Die Betriebssystemversion, mit der Sie arbeiten.
- Die aktuellen Zeit- und Datumsangaben.

Informationen über den globalen Speicher

TDW gibt Ihnen folgende Informationen über den globalen Speicher:

Mode	Folgende Speicher-Modi sind möglich: Large-frame EMS, Small-frame EMS und non-EMS (Extended Memory).
Banked	Speicher in KByte oberhalb der EMS *Bank Line* (der zum Expanded Memory werden kann, wenn das System dieses verwendet).
Not banked	Speicher in KByte unterhalb der EMS *Bank Line* (der nicht zum Expanded Memory werden kann).
Largest	Der größte zusammenhängende freie Speicherbereich in KByte.

Meldungen in der Statuszeile

Folgende Meldungen in der zweiten Zeile (Statuszeile) geben den Grund an, warum das Programm angehalten wurde:

Stopped at__
[Angehalten in__]
Ihr Programm wurde nach Ausführung der folgenden Befehle angehalten: *Run /Execute to*, *Run/Go to cursor* oder *Run/Until return*. In der Statuszeile wird außerdem dargestellt, wann das Programm zum ersten Mal geladen wurde und daß Ihr Programm den Startup-Code des Compilers bis zum Anfang Ihres Quellcode-Moduls ausgeführt hat.

No program loaded
[Kein Programm geladen]
Sie haben TDW ohne ein Programm gestartet. Sie können keinen Code ausführen, bis Sie entweder ein Programm laden oder einige Befehle mit dem lokalen Menübefehl *Assemble* im Code-Ausschnitt des CPU-Fensters assemblieren.

Trace
[Einzelschritt ausgeführt]
Sie haben eine einzelne Quellzeile oder einen Maschinenbefehl mit F7 oder *Run/Trace* ausgeführt.

Step
[Einzelschritt ausgeführt]
Sie haben einen einzelnen Befehl oder einen Maschinenbefehl mit F8 oder dem Befehl *Run/Step* ausgeführt, bei dem Unterprogrammaufrufe übersprungen werden.

Breakpoint at __
[Breakpoint bei __]
Ihr Programm ist an einem Breakpoint angehalten worden. Hinter »at« steht die Programmadresse des Breakpoints.

Window message breakpoint at __
[Fensterbotschafts-Breakpoint bei __]
Ihr Programm ist an einem Fensterbotschafts-Breakpoint angehalten worden. Hinter »at« steht die Fenster-Prozedur, für die die Meldung bestimmt war.

Terminated, exit code __
[Beendet, Exit-Code __]
Das Programm ist bis zum Ende ausgeführt worden. Hinter »code« steht der Exit-Code, der von Ihrem Programm an Windows zurückgegeben wird. Wenn Ihr Programm keinen bestimmten Wert zurückgibt, erhalten Sie ein zufälliges Ergebnis. Sie können Ihr Programm nicht mehr starten, bis Sie das Programm mit *Run/Program reset* erneut laden.

Loaded
[Geladen]
Sie haben TDW mit einem Programm aufgerufen. *Au331erdem* haben Sie eine Kommandozeilenoption angegeben, mit der verhindert wird, daß der Startup-Code des Compilers ausgeführt wird. Zu diesem Zeitpunkt wurden noch keine

Befehle, auch nicht zur Initialisierung des Stack oder des Segmentregisters, ausgeführt. Das bedeutet, daß Sie bei der Untersuchung von Programmdaten falsche Werte vorfinden werden.

Interrupt
[Interrupt]
Sie haben die Interrupt-Taste (Strg Alt S-Abf) gedrückt, um wieder die Steuerung des Programms zu übernehmen. Ihr Programm wird sofort unterbrochen, und der Debugger wird aktiviert.

Exception __
[Ausnahme __]
Eine Ausnahmebedingung des Prozessors ist aufgetreten. Das passiert meistens, wenn in Ihrem Programm ein unzulässiger Befehl ausgeführt werden soll. In den Intel-Handbüchern sind die entsprechenden Ausnahmebedingungen ausführlich dokumentiert.

Im Zusammenhang mit Windows-Programmen triff am häufigsten Exception 13 auf; dadurch wird angezeigt, daß Ihr Programm einen unzulässigen Speicherzugriff versucht hat. (Entweder ist der Selector-Wert in einem Segmentregister falsch oder der Offset-Teil einer Adresse weist über die Segmentgrenze hinaus.) Sie müssen den falschen Zeiger, der das Problem verursacht, berichtigen.

Divide by zero
[Division durch Null]
In Ihrem Programm wurde versucht, eine Division durch Null durchzuführen.

Global breakpoint __ at __
[Globaler Breakpoint __ an Position __]
Es wurde ein globaler Breakpoint ausgelöst. Sie erhalten die Nummer des Breakpoint und die Stelle im Programm, an der der Breakpoint auftrat.

Das Menü Run

Das Menü *Run* enthält eine Reihe von Optionen, mit denen Sie verschiedene Abschnitte Ihres Programms ausführen können. Da diese Optionen sehr häufig gebraucht werden, sind sie alle über Funktionstasten abrufbar.

Run

F9

Durch den Befehl *Run* (engl. starten) wird das Programm mit der vollen Ablaufgeschwindigkeit ausgeführt. Der Debugger erhält erst dann wieder die Steuerung, wenn eines der folgenden Ereignisse eintritt:

- das Programm wird beendet;
- ein Breakpoint wird erreicht;
- Sie unterbrechen das Programm mit Strg Untbr S-Abfr .

Go to cursor F4

Mit dem Befehl *Go to cursor* (engl. Gehe bis zum Cursor) wird Ihr Programm bis zu der Zeile ausgeführt, in der der Cursor im aktuellen *Module*- oder *CPU*-Fenster steht. Wenn gerade ein *Module*-Fenster aktiv ist, muß der Cursor auf einer Quellzeile in einer Funktion stehen.

Trace into F7

Eine einzelne Quellzeile oder einen Maschinensprachebefehl kann man mit dem Befehl *Trace into* (engl. nachspüren) ausführen. Normalerweise wird eine einzelne Quellzeile ausgeführt. Wenn die aktuelle Zeile einen Prozedur- oder Funktionsaufruf enthält, führt TDW die Befehle in dieser Routine schrittweise aus. Falls jedoch ein *CPU*-Fenster aktiv, wird nur ein einziger Assembler-Befehl ausgeführt.

TDW behandelt Objektmethoden und Elementfunktionen einer Klasse wie jede andere Prozedur oder Funktion. Mit F7 werden die Befehle des Quellcodes schrittweise ausgeführt.

Step over F8

Der Befehl *Step over* (engl. schrittweise überspringen) führt eine einzelne Quellzeile oder einen Maschinenbefehl aus, übergeht jedoch alle Prozedur- und Funktionsaufrufe. In einem *Module*-Fenster wird normalerweise eine Quellzeile ausgeführt. In einem *CPU*-Fenster wird durch F8 bei einer CALL-Anweisung das aufgerufene Unterprogramm vollständig ausgeführt.

Wenn Sie über eine einzelne Quellzeile springen, behandelt TDW alle Funktions- oder Prozeduraufrufe in dieser Zeile als Bestandteil der Zeile, und Sie landen nicht am Anfang einer dieser Funktionen. Stattdessen kommen Sie direkt zur nächsten Zeile in der aktuellen Routine bzw. in der Routine, die diese Routine aufgerufen hat.

Wenn Sie einen einzelnen Maschinenbefehl überspringen, interpretiert TDW bestimmte Befehle als Einzelbefehle, auch wenn damit mehrere Befehle auf einmal ausgeführt werden. Die folgende Aufstellung enthält alle Befehle, die TDW mit *Step over* als einen einzelnen Befehl verarbeitet:

- `CALL` Unterprogrammaufruf, `NEAR` oder `FAR`

- `INT` Interrupt-Aufruf
- `LOOP` Schleife mit CX als Zähler
- `LOOPZ` Schleife mit CX als Zähler
- `LOOPNZ` Schleife mit CX als Zähler

Außerdem werden auch alle Kombinationen von `REP`, `REPNZ` oder `REPZ` mit `CMPS`, `CMPSB`, `CMPSW`, `LODS`, `LODSB`, `LODSW`, `MOVS`, `MOVSB`, `MOVSW`, `SCAS`, `SCASB`, `SCASW`, `STOS`, `STOSB` und `STOSW` als Einzelbefehl behandelt.

Der Befehl *Run/Step over* behandelt alle Objektmethoden oder Elementfunktionen einer Klasse als Einzelbefehl und übergeht sie wie alle anderen Prozedur- und Funktionsaufrufe.

Execute to... Alt F9

Mit dem Befehl *Execute to* (engl. ausführen bis) wird das Programm bis zur angegebenen Adresse ausgeführt. Die Adresse wird unter Umständen gar nicht erreicht, wenn vorher ein Breakpoint ausgelöst oder das Programm von Ihnen unterbrochen wurde.

Until return Alt F8

Nach der Eingabe des Befehls *Until return* (engl. bis zum Rücksprung) wird das Programm solange ausgeführt, bis die aktuelle Funktion zur aufrufenden Funktion zurückkehrt. Das ist vor allem in folgenden Situationen sehr praktisch: wenn Sie mit *Run/Trace* aus Versehen die schrittweise Abarbeitung einer Prozedur oder Funktion begonnen haben, die Sie eigentlich gar nicht interessiert, oder wenn Sie festgestellt haben, daß die aktuelle Funktion zu Ihrer Zufriedenheit funktioniert und Sie den Rest gar nicht mehr überprüfen wollen.

Animate...

Mit dem Befehl *Animate* (engl. animieren) wird das Programm langsam Schritt für Schritt ausgeführt, wobei alle Veränderungen nach jedem einzelnen Programmschritt dargestellt werden. Damit kann man die aktuelle Position im Quellcode und die Variablenwerte verfolgen. Dieser Befehl kann mit einem beliebigen Tastendruck unterbrochen werden.

Nachdem Sie *Run/Animate* ausgewählt haben, werden Sie nach der gewünschten Verzögerungsdauer zwischen zwei Programmschritten gefragt. Die Zeitverzögerung wird in Zehntelsekunden gemessen; der Standardwert beträgt 3.

Back trace

Wenn Sie ein Programm schrittweise ausführen (F7 oder Alt F7) können Sie mit diesem Befehl die letzten Befehle rückwärts ausführen. Dies ist sehr praktisch, wenn Sie über den Punkt hinausgegangen sind, an dem Sie einen Fehler vermuten. Damit können Sie die Ausführung des Programms ungeschehen machen. Dies kann ein einzelner Programmschritt oder eine bestimmte Stelle sein, die im Befehlsausschnitt des Fensters *Execution History* ausgewählt ist.

Achtung: Es sind einige Einschränkungen zu beachten (siehe »Der Ausschnitt mit den ausgeführten Programmschritten« weiter hinten in diesem Kapitel).

Die Möglichkeit, ein Programm rückwärts auszuführen, ist im CPU-Fenster immer gegeben. Damit der Quellcode jedoch von hinten her abgearbeitet werden kann, muß die History-Aufzeichnung aktiviert sein; öffnen Sie dazu mit *View/Execution History* das entsprechende Fenster, und aktivieren Sie im lokalen Menü die History-Aufzeichnung (Full History *On*).

TDW kann Windows-Code, der von Ihrem Programm aufgerufen wurde, nur dann rückwärts ausführen, wenn gerade ein *CPU*-Fenster aktiv ist und der entsprechende Programmcode aus einer DLL stammt, die zum Debuggen ausgewählt worden ist.

Instruction trace Alt F7

Der Befehl *Instruction trace* (engl. Befehlsschritt) führt einen einzelnen Maschinenbefehl aus. Verwenden Sie diese Option, wenn Sie einen Interrupt nachvollziehen wollen oder wenn Sie eine Prozedur oder Funktion, die keine Debug-Informationen enthält (zum Beispiel eine Bibliotheksroutine), aus einem *Module*-Fenster heraus debuggen wollen.

Da Sie sich nicht mehr am Anfang einer Quellzeile befinden, wird automatisch ein *CPU*-Fenster geöffnet.

Arguments...

Mit diesem Befehl können Sie andere Kommandozeilenargumente für Ihr Programm angeben (siehe »Die Argumente eines Programms ändern« am Ende dieses Kapitels).

Program reset Strg F2

Der Befehl *Program reset* (engl. Programm zurücksetzen) lädt das Programm, das Sie debuggen, erneut. Verwenden Sie diesen Befehl,

- wenn Sie mit der Ausführung des Programms »zu weit« gegangen sind (d.h., wenn Sie die Fehlerstelle übersprungen haben).

- wenn Ihr Programm beendet ist und Sie es noch einmal starten möchten.

- bei einer Windows-Anwendung, wenn Sie sich in einem *Module*- oder *CPU*-Fenster befinden, die Anwendung mit Strg Alt S-Abf angehalten haben und diese jetzt beenden und neu starten möchten.

- wenn Sie Ihre Windows-Anwendung schon geladen und Startup-Debuggen für eine oder mehrere dynamische Link-Bibliotheken (DLLs) eingestellt haben und Sie jetzt diese DLLs debuggen möchten.

Wenn Sie sich in einem *Module*- oder *CPU*-Fenster befinden, wird die Zeilenmarkierung auf den Anfang des Programms gesetzt, der Programmstart wird jedoch nicht dargestellt. Stattdessen bleiben Sie genau an der Stelle, an der Sie vor dem Befehl *Program reset* waren. Dadurch wird es für Sie leichter, den Cursor dorthin zu positionieren, wo Sie vorher waren, und das Programm bis zu dieser Zeile laufen zu lassen.

Wenn Sie den Befehl *Program reset* ausgewählt haben, weil Sie gerade eine Anweisung in Ihrem Quelltext zuviel ausgeführt haben, können Sie den Cursor ein paar Zeilen im Quelltext zurücksetzen und F4 drücken, um das Programm bis zu dieser Position laufen zu lassen. Alternativ könnten Sie, anstatt *Program Reset* zu wählen, mit *Run/Backtrace* den vorher ausgeführten Code schrittweise rückwärts ausführen, wenn Sie *Full History* (das lokale Fenster des Fensters *View/Execution History*) einstellen.

Das Fenster Execution History

TDW bietet eine ganz spezielle Möglichkeit der Aufzeichnung aller ausgeführten Programmschritte in einem Fenster: die *Execution history*. In diesem Fenster können Sie außerdem alle Tastatureingaben erfassen, die Sie bis zu einer bestimmten Stelle im Programm eingegeben haben. Diese Anweisungen können zum einen untersucht werden und zum anderen dazu verwendet werden, an eine bereits übersprungene Programmstelle zurückzugehen, an der Sie einen Fehler vermuten. TDW kann etwa bis zu 400 Anweisungen aufzeichnen.

Sie können die Programmschritte im Fenster *Execution history* untersuchen, das sich mit dem Befehl *View/Execution history* öffnen läßt.

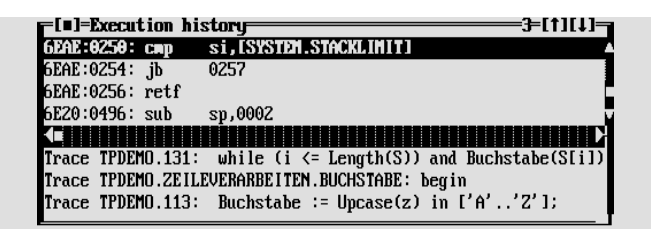


Abbildung 5.5
Das Fenster Execution History

Der Ausschnitt mit den ausgeführten Programmschritten

Der Befehlsausschnitt zeigt die Befehle, die bereits ausgeführt wurden und untersucht oder ungeschehen gemacht werden können. Benutzen Sie den Auswahlbalken zur Selektion.

Hinweis: Im Ausschnitt *Execution history* werden nur die Programmschritte aufgezeichnet, die mit dem Befehl *Trace into* (F7) oder *Instruction Trace* (Alt F7) ausgeführt wurden. Außerdem werden die *Step over*-Befehle festgehalten, solange keiner der Befehle ausgeführt wird, die im Rahmen dieses Befehls angegeben sind. Sobald Sie einen *Run*-Befehl ausführen oder einen Interrupt aufrufen, wird die *Execution history* zerstört. (Die Aufzeichnung beginnt wieder ab dem nächsten *Trace*-Befehl).

Achtung: Interrupt-Aufrufe lassen sich nicht zurückverfolgen. Wenn Sie über einen Prozedur- oder Funktionsaufruf hinausgelangen, ist der Befehl vor dem Rücksprung nicht mehr erreichbar. Sie können auch nicht über Befehle zurückgehen, die I/O-Ports benutzen, da Schreib- und Lesebefehle dort nicht mehr rückgängig gemacht werden können.

Das lokale Menü des Befehlsausschnitts

Im lokalen Menü des Befehlsausschnitts stehen drei Befehle: *Inspect*, *Reverse Execute*, *Full History*.

Inspect Dieser Befehl führt Sie zu einem Befehl, der im Befehlsausschnitt hervorgehoben dargestellt ist. Wenn es sich um eine Quellzeile handelt, wird die entsprechende Zeile im *Module*-Fenster angezeigt. Falls kein Quellcode verfügbar ist, wird das *CPU*-Fenster geöffnet, in dem der Befehl im Code-Ausschnitt angezeigt wird.

Reverse Execute Mit diesem Befehl können Sie das Programm rückwärts bis zu der Programmstelle ausführen, die im Befehlsausschnitt hervorgehoben dargestellt ist. Wenn Sie eine Quellcodezeile ausgewählt haben, erscheint das *Module*-Fenster, anderenfalls das *CPU*-Fenster, in dem der Programmschritt im Code-Ausschnitt hervorgehoben wird. (Das Tastenkürzel für diesen Befehl lautet Alt F4 .)

Achtung: Ein Programmabschnitt, der nicht per Einzelschritt abgearbeitet wurde, kann auch nicht rückwärts ausgeführt werden. Wenn Sie zum Beispiel einen Breakpoint gesetzt haben und das Programm mit F9 bis zu dieser Programmstelle ausgeführt haben, wird Ihre gesamte *Execution history* verworfen. In diesem Fall bleibt Ihnen zum Wiederherstellen der vorherigen Situation nur noch das Abspielen der Tastatureingaben, indem Sie das Programm erneut laden und bis zur gewünschten Stelle ausführen.

Durch die Ausführung eines `INT`-Befehls werden ebenfalls alle Aufzeichnungen in der *Execution history* verworfen. Über diesen Befehl kommen Sie nicht zurück, falls Sie die Ausführung des Interrupt nicht mit Alt F7 begonnen haben. Durch die folgenden Befehle werden zwar die Aufzeichnungen nicht gelöscht, ihre Wirkung kann jedoch nicht ungeschehen gemacht werden. Sie

sollten auf Überraschungen gefaßt sein, wenn Sie solche Befehle rückwärts ausführen wollen.

```
IN      INSW
OUT     OUTSB
INSB    OUTSW
```

Full history Mit diesem Befehl können Sie die Rückwärts-Ausführung des Programms ein- oder ausschalten.

Programmausführung unterbrechen

Windows-Anwendungen sind interaktive Programme; ein solches Programm untersucht man am besten, indem man es einfach startet und dann unterbricht, oder dafür sorgt, daß es bei der Ausführung auf einen Breakpoint stößt.

Beim Debuggen einer Windows-Anwendung ist die schrittweise Abarbeitung der Anwendung als grundsätzliche Vorgehensweise unter Umständen nicht sehr hilfreich, da Sie möglicherweise auf einen Code-Bereich stoßen, der sich in einer Schleife befindet und auf die Botschaft eines Fensters wartet; deshalb sollten Sie nach Möglichkeit Breakpoints für Code- und Fensterbotschaften setzen und Ihr Programm ausführen, bis es auf einen Breakpoint stößt. Anschließend arbeiten Sie es schrittweise durch.

Wenn Ihr Programm in einer Schleife auf eine Botschaft wartet, können Sie sich die Ausgaben Ihres Programms ansehen, indem Sie die Alt F5 eingeben; es ist dann allerdings nicht möglich, mit dem Programm in Wechselwirkung zu treten. Starten Sie stattdessen das Programm mit F9, so daß Sie die Fenster des Anwendungsprogramms benutzen können. Was ist aber zu tun, wenn es nötig wird, zu TDW zurückzukehren, um einen Fehler zu lokalisieren, der in einem der Fenster Ihres Programms aufgetreten ist?

Sie können das Programm durch Eingabe der Tastenkombination Strg Untbr S-Abfr unterbrechen. Wenn TDW daraufhin die Steuerung wieder übernommen hat, können Sie Code- oder Botschafts-Breakpoints setzen, Botschaften ansehen oder weitere notwendige Schritte durchführen, um den Fehler aufzuspüren. Wenn Sie alle nötigen Maßnahmen getroffen haben, können Sie Ihr Programm durch Betätigen von F9 wieder starten.

Achtung: Wenn TDW die Steuerung übernommen hat, und Sie ein *CPU*-Fenster ohne Zeilen vorfinden, die Ihrem Code entsprechen, haben Sie vermutlich einen Bereich mit Windows-Code vor sich. Sie können das *Module*-Fenster öffnen und Breakpoints setzen oder sonstige Schritte unternehmen; es gibt jedoch einige Dinge, die Sie nicht tun sollten:

- Arbeiten Sie das Programm nicht im Einzelschrittmodus durch. Wenn Sie den Einzelschrittmodus versuchen, nachdem Sie Ihre Anwendung unterbrochen haben, kann es für den Fall, daß gerade Windows-Code ausgeführt wurde, zu Fehlern kommen. Normalerweise beendet Windows in einer solchen Situation

sowohl das Anwendungsprogramm als auch TDW; dabei wird die Meldung »Unrecoverable application error« (nicht behebbarer Fehler in der Anwendung) ausgegeben.

■ Sie dürfen weder Ihr Anwendungsprogramm noch TDW beenden oder erneut laden. Falls Sie das tun, stürzt TDW ab. Wenn Sie in einer solchen Situation versuchen, das Programm zu verlassen oder erneut zu laden, gibt TDW in einem Dialogfenster die folgende Eingabeaufforderung aus:

```
Ctrl-Alt-SysRq interrupt, system crash possible, Continue?
```

Es ist hier in jedem Fall angebracht, mit *No* zu antworten, zu TDW zurückzukehren und einen Breakpoint zu setzen, bei dem Sie wissen, daß er auf jeden Fall beim Ablauf des Programm erreicht wird; starten Sie das Programm anschließend erneut, sorgen Sie dafür, daß es auf den Breakpoint trifft und wechseln Sie dann zu TDW.

Programm beenden

Wenn Ihr Programm beendet wird, übernimmt der Debugger wieder die Steuerung. Er zeigt Ihnen den Exit-Code, mit dem Ihr Programm zu Windows zurückkehrt. Nachdem Ihr Programm beendet wurde, können Sie die Optionen des *Run*-Menüs nur dann wieder verwenden, falls Sie Ihr Programm mit *Run/Program reset* erneut laden.

Die Segmentregister und der Stack sind normalerweise nicht mehr in Ordnung, wenn Ihr Programm beendet wurde. Sie sollten also nach dem Programmende keine Programmvariablen mehr untersuchen oder verändern.

Eine Debuggersitzung erneut starten

TDW bietet einige Möglichkeiten, mit deren Hilfe man eine Debugger-Sitzung so einfach wie möglich erneut starten kann. Beim Debuggen passiert es leicht, daß man etwas zu weit geht und über das Ziel hinausschießt. Sie müssen den Debugger dann erneut starten und mit der Ausführung des Programms vor den letzten Befehlen aufhören, bei denen Sie außer Tritt gekommen sind.

Bei den meisten Debuggern sind Sie dazu gezwungen, alle Befehle noch einmal von Hand einzugeben, um zur entsprechenden Stelle im Programm zu kommen. Das kann sehr mühsam sein. TDW hat eine sehr leistungsfähige Option, mit der Sie die Tastatureingaben der letzten Sitzung aufzeichnen und bei Bedarf wieder abspielen können. Man kann damit auch wieder das letzte Programm mit den entsprechenden Kommandozeilenoptionen laden.

Programm erneut laden

Um das Programm wieder zu laden, das Sie gerade bearbeitet haben, wählen Sie den Befehl *Run/Program reset* . Turbo Debugger lädt dann das Programm mit allen Daten, die Sie seit der letzten Abspeicherung hinzugefügt haben. Das ist der sicherste Weg, um ein Programm erneut zu starten. Sehr riskant ist es dagegen, den Cursor einfach an den Anfang des Programms zu setzten und es von dort aus zu starten, da viele Programme voraussetzten, daß Daten beim Programmstart mit bestimmten Werten initialisiert sind. Beachten Sie, daß beim Befehl *Program reset* alle Breakpoints und *Watches* -Ausdrücke erhalten bleiben.

Ein neues Programm zum Debuggen laden

Mit Hilfe des Befehls *File/Open* können Sie ein neues Programm zum Debuggen laden.

Sie können einen Dateinamen (mit der Erweiterung »EXE«) im Eingabefeld eingeben oder CR drücken, um eine Liste aller »EXE«-Dateien des aktuellen Verzeichnisses zu erhalten. Positionieren Sie den Auswahlbalken auf die Datei, die Sie laden wollen und drücken Sie CR

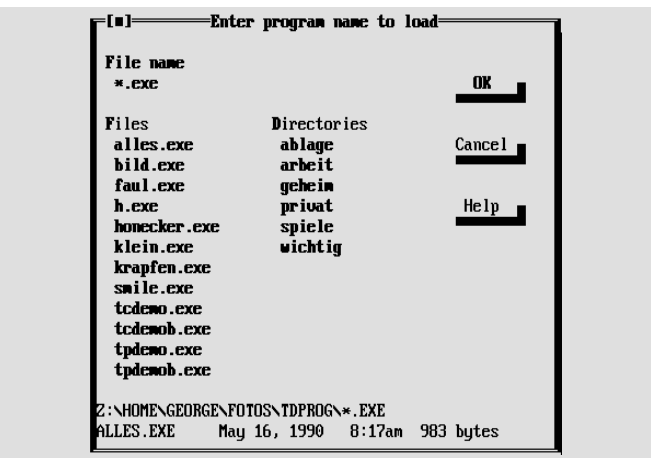


Abbildung 5.6 Dialogfenster Load Program zum Laden von Programmen

Wenn Sie stattdessen den Namen der gewünschten Datei eingeben, bewegt sich der Auswahlbalken zu der Datei, die mit dem/den ersten eingebenen Buchstaben beginnt. Wenn der Auswahlbalken auf der gewünschten Datei steht, drücken Sie CR

Sie können zusätzliche Argumente für das Programm, das Sie debuggen wollen, hinter dem Programmnamen eingeben:

```
myprog a b c
```

Damit wird das Programm `MyProgy` mit drei Kommandozeilenparametern, nämlich `a`, `b` und `c` geladen.

Die Argumente eines Programms verändern

Wenn Sie beim Laden Ihres Programms nicht die richtigen Argumente verwendet haben, dann können Sie mit dem Befehl *Run/Arguments* fehlende oder falsche Argumente eingeben bzw. verändern. Geben Sie die neuen Argumente genauso ein, wie Sie sie etwa in einer DOS-Kommandozeile hinter dem Programmnamen angeben würden.

Sobald Sie die neuen Argumente eingegeben haben, fragt Sie TDW, ob Sie Ihr Programm noch einmal laden wollen. Sie sollten diese Frage normalerweise immer mit *Yes* (engl. ja) beantworten, da die Argumente bei den meisten Programmen erst nach dem Laden des Programms verfügbar sind.

6. Programmdaten untersuchen und ändern

TDW bietet einmalige Möglichkeiten, um intuitiv auf die Daten eines Programms zuzugreifen:

- Mit *Inspector*-Fenstern können Sie Daten in der Art und Weise untersuchen, wie sie in Ihrer Quelldatei auftreten. Sie können Zeigern »folgen«, Arrays durchsuchen und Strukturen, Records und Varianten so betrachten, wie Sie sie in Ihre Quelldatei geschrieben haben.

- Variablen und Ausdrücke können in *Watches* -Fenstern untergebracht werden, wo Sie die Werte während der Programmausführung ständig kontrollieren können.

- Im Dialogfenster *Evaluate/Modify* können Sie sich den Inhalt jeder Variablen anzeigen lassen und einen neuen Wert zuweisen.

In diesem Kapitel wird davon ausgegangen, daß Sie die Datenstrukturen der Programmiersprache, die Sie verwenden (Turbo C++ for Windows), verstehen. Wenn Sie noch nicht viel mit der Programmiersprache gearbeitet haben und noch nicht alle Datentypen kennen, dann erhalten Sie in diesem Kapitel genügend Informationen über die einfachen Datentypen (`char`, `int`, `integer`, `boolean`, `real`, Fließkommazahlen einfacher und doppelter Genauigkeit, Strings, `long integer`, usw.). Wenn Sie sich in die komplizierteren Datentypen (Zeiger, Records, Strukturen, Varianten usw.) vertieft haben, dann sollten Sie zu diesem Kapitel zurückkehren, um mehr darüber zu erfahren, wie diese Datentypen mit TDW betrachtet werden können.

In diesem Kapitel wird gezeigt, wie Variablen in Ihrem Programm untersucht und verändert werden können. Zunächst erläutern wir das Menü *Data* und die darin enthaltenen Befehle. Daraufhin besprechen wir, wie man Programmdaten mit Ausdrücken, die Seiteneffekte aufweisen, ändern kann. Anschließend wird dargestellt, wie man auf Datenobjekte in Ihren Quellmodulen direkt zugreifen kann. Wir stellen das *Watches* -Fenster vor und beschreiben zum Abschluß die Darstellung einfacher Datentypen für jede einzelne Sprache in einem *Inspector*-Ausschnitt.

Wenn Sie einen beliebigen Speicherbereich als Hex-Daten untersuchen oder verändern wollen, sollten Sie in Kapitel 12, »Debuggen auf Assembler-Ebene«, weiterlesen. Dort wird das Debuggen auf Assembler-Ebene dargestellt.

Das Menü Data

Im *Data*-Menü kann man Programmdaten untersuchen und verändern. Sie können dort auch Ausdrücke auswerten, die den Wert von Variablen verändern und *Inspector*-Fenster öffnen, um den Inhalt Ihrer Daten darzustellen.

Inspect...

Nach dem Befehl *Inspect* werden Sie um eine Variable gebeten, die die von Ihnen gewünschten Daten enthält. Anschließend wird ein *Inspector*-Fenster geöffnet, das den Inhalt der Variablen oder des Ausdrucks anzeigt. Sie können einen einfachen Variablennamen oder auch einen komplexen Ausdruck angeben. Der Ausdruck muß sich lediglich auf einen Speicherbereich beziehen und darf nicht einfach nur eine Konstante ergeben.

Wenn sich der Cursor zum Zeitpunkt des Befehls im Textausschnitt befindet, erhalten Sie im Dialogfenster sofort den Namen der Variablen, auf der der Cursor steht. Wenn Sie einen Ausdruck mit der Taste Einfg ausgewählt haben, dann enthält die Eingabeaufforderung den selektierten Ausdruck.

Die *Inspector*-Fenster werden erst dann wirklich interessant, wenn es um komplizierte Datenstrukturen geht, zum Beispiel um ein Array oder um eine verkettete Liste. Da Sie die Objekte in einem *Inspector*-Fenster genau untersuchen können, können Sie ebenso leicht durch die Datenstrukturen in Ihrem Programm »wandern« wie durch Ihren Quelltext im *Module*-Fenster.

Hinweis: Sehen Sie im nachfolgenden Abschnitt, »Inspector-Fenster«, nach, welche Möglichkeiten die *Inspector*-Fenster bieten.

Evaluate/ Modify...

Beim Befehl *Evaluate/Modify* werden Sie um einen Ausdruck gebeten, der ausgewertet werden soll. Dieser Ausdruck wird dann genauso ausgewertet wie vom Compiler. In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Erörterung der Ausdrücke.

Wenn sich der Cursor bei diesem Befehl in einem Textausschnitt befindet, so wird Ihnen die Variablenbezeichnung angeboten, auf der der Cursor gerade steht. Haben Sie einen Ausdruck mit Einfg ausgewählt, wird Ihnen der markierte Ausdruck zur Übernahme angeboten.

Vergessen Sie nicht, daß Sie eine Formatangabe hinter dem Ausdruck, den Sie auswerten wollen, angeben können. TDW stellt das Ergebnis des Datentyps dar. Um das Ergebnis in einem anderen Format darzustellen, schreiben Sie ein Komma nach dem Ausdruck und dann eine Formatangabe. In Kapitel 9, »Ausdrücke«, finden Sie eine Beschreibung der Formatsteuerung. Solche Angaben sind sehr praktisch, wenn man etwas beobachten will, aber als Format nicht die Standarddarstellung von TDW verwendet werden soll.

Das Dialogfenster ist in drei Eingabefelder aufgeteilt:

Abbildung 6.1 Das Dialogfenster Evaluate/Modify

- In das obere Eingabefeld können Sie den Ausdruck eingeben, der ausgewertet soll. Dies ist das Eingabefeld *Evaluate* . Es enthält, wie andere Eingabefelder, eine History-Liste.

- Das mittlere Eingabefeld stellt das Ergebnis der Auswertung dar.

- Im unteren Eingabefeld können Sie einen neuen Ausdruck eingeben. Wenn der Ausdruck nicht verändert werden kann, steht in diesem Ausschnitt »Not available« (engl. nicht verfügbar), und Sie können den Cursor nicht in diesen Ausschnitt bewegen.

Ihre Eingaben in den Eingabefeldern *Evaluate* oder *New value* werden nach Auswahl des Befehlsfelds *Modify* ausgewertet. Zwischen den einzelnen Eingabefeldern können Sie wie in anderen Dialogfenstern mit den Tasten Tab und Umschalt Tab umschalten. Wenn Sie in einem der Ausschnitte Esc drücken oder das Feld *Cancel* auswählen oder mit der Maus anklicken, wird das Dialogfenster entfernt.

Datenstrings, die zu lang für die Ausgabe im *Result*-Feld sind, werden rechts durch einen Pfeil abgeschnitten. Den Rest der Ausgabe können Sie durch Rollen nach rechts einblenden.

Wenn Sie ein Programm debuggen, das in Turbo C++ for Windows verfaßt wurde, können Sie im Dialogfenster *Evaluate/Modify* die Felder einer Objektinstanz oder die Elemente einer Klasseninstanz untersuchen. Sie können für die Ausgabe der Instanzen alle für Records zulässigen Formatangaben verwenden.

Falls Sie sich innerhalb einer Elementfunktion befinden, berücksichtigt TDW den Gültigkeitsbereich des `this`-Parameters und verwendet automatisch die entsprechenden Formatangaben.

Mit TDW können Sie auch eine Elementfunktion innerhalb des Dialogfensters *Evaluate/Modify* aufrufen. Sie brauchen nur den Namen der Instanz gefolgt von einem Punkt und dahinter den Namen der Elementfunktion sowie die gewünschten Parameter einzugeben. (Konstruktoren und Destruktoren lassen sich im *Evaluate* -Fenster jedoch nicht auswerten.)

Nehmen wir einmal an, es liegen folgende Deklarationen vor:

```
class point {
  public:
    int x, y, visible;

    point ();
    ~point ();
    int Show ();
    int Hide ();
    void MoveTo (int NewX, int NewY);
}

  point APoint;
```

Sie könnten dann folgende Ausdrücke im *Evaluate* -Feld eingeben und würden die nebenstehenden Ergebnisse erhalten:

Ausdruck	Ergebnis
`APoint.xint 2 (0x2)`	
`APointclass point {1,2,27489}`	
`APoint.MoveTovoid () @6B61:0299`	
`APoint.Showint () @6B61:0285`	
`APoint.Show ()`	`int 1 (0x1)`

Ausdrücke mit Seiteneffekten Die Programmiersprache C bietet eine Möglichkeit, die man als Ausdruck mit Seiteneffekten bezeichnen könnte, die sehr praktisch und leistungsfähig ist, ab und zu aber auch einige Überraschungen birgt.

Ein Ausdruck mit Seiteneffekten verändert den Wert von einer oder mehreren Variablen, während der Ausdruck ausgewertet wird. Die C-Operatoren für Inkrement (`++`) und Dekrement (`--`) sowie die Zuweisungsoperatoren (`=`, `+=` usw.) bewirken zum Beispiel Seiteneffekte. Wenn Sie Funktionen in Ihrem Programm innerhalb eines C-Ausdrucks aufrufen (wie zum Beispiel in `funky(2)`), dann müssen Sie damit rechnen, daß Ihre Funktion unerwartete Seiteneffekte aufweisen kann.

Wenn Sie den Wert einer Variablen nicht verändern, sondern lediglich einen Ausdruck auswerten wollen, der einige Ihrer Programmvariablen enthält, sollten Sie keine Operatoren mit Seiteneffekten verwenden. Auf der anderen Seite kann man mit Seiteneffekten sehr schnell den Wert einer Variablen oder eines Speicherbereichs verändern. Wenn Sie zum Beispiel den Wert Eins zum Wert der Variablen `Zaehler` hinzuaddieren wollen, so brauchen Sie nur den C-Ausdruck `Zaehler++` auszuwerten.

Sie können das Eingabefeld *Evaluate /Modify* auch als einfachen »Taschenrechner« verwenden, indem Sie als Operanden Zahlenwerte statt Programmvariablen eingeben.

Add Watch...

Beim Befehl *Add Watch* (engl. beobachten) werden Sie um eine Anweisung oder eine Programmvariable gebeten, die in die Liste der Variablen im *Watches* -Fenster aufgenommen werden soll.

Wenn der Cursor bei diesem Befehl in einem Textausschnitt steht, dann enthält das Dialogfenster automatisch die Variable unter dem Cursor. Wenn Sie einen Ausdruck mit Einfg selektieren, wird das Eingabefeld im Dialogfenster mit dem selektierten Ausdruck initialisiert.

Function Return

Mit dem Befehl *Function Return* können Sie den Wert abfragen, den die Funktion zum Zeitpunkt der Auswertung als Ergebnis zurückliefern würde. Sie sollten diesen Befehl kurz vor der Rückkehr zur aufrufenden Funktion verwenden.

Der Rückgabewert wird in einem *Inspector*-Fenster angezeigt, so daß Sie auch die Rückgabewerte von Zeigern und zusammengesetzten Datenobjekten bequem untersuchen können.

Durch diesen Befehl ist es nicht mehr nötig, in das *CPU*-Fenster umzuschalten, um den Registern den Rückgabewert zu entnehmen. Da zudem bekannt ist, welcher Datentyp zurückgeliefert werden soll, und Wert daher korrekt formatiert ausgegeben werden kann, ist dieser Befehl viel einfacher als ein Hex-Dump anzuwenden.

Auf Datenobjekte in Quelldateien zeigen

TDW hat einen leistungsfähigen Mechanismus, der Ihnen viel Tipparbeit beim Eingeben der Variablennamen erspart, wenn Sie eine Variable untersuchen wollen. In einem *Module*-Fenster können Sie den Cursor irgendwo auf einen Variablennamen positionieren und dann den lokalen Menübefehl *Inspect* eingeben, um ein *Inspector*-Fenster zur Anzeige des Variableninhalts zu erzeugen. Sie können auch einen Ausdruck, der untersucht werden soll, selektieren, indem Sie Einfg drücken und den Ausdruck mit den Cursortasten hervorheben, bevor Sie den Befehl *Inspect* auswählen. (In Kapitel 8, »Dateien untersuchen und ändern«, finden Sie eine ausführliche Behandlung der *Module*-Fenster.)

Das Fenster Watches

Im *Watche* s-Fenster kann man Variablen und Ausdrücke aus einem Programm auflisten, die man dauernd beobachten will. Sie können den Wert von einfachen Variablen, wie zum Beispiel einer Integer-Variablen, beobachten, aber auch komplexe Datenstrukturen überwachen – wie ein Array. Zudem kann man den Wert eines berechneten Ausdrucks, der sich nicht nur auf eine Speicherstelle bezieht, im Auge behalten, wie zum Beispiel `x * y + 4`.

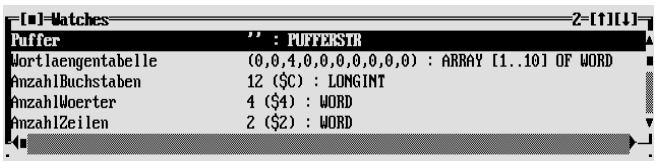


Abbildung 6.2
Das Watches-Fenster

Wählen Sie den Befehl *View/Watches* aus, um ein Watches-Fenster zu erzeugen. Das Fenster stellt eine Liste von Variablen oder Ausdrücken dar, deren Werte Sie überwachen wollen. Bei allen Objekten wird der Variablenname oder der Ausdruck auf der linken Seite angegeben, der Datentyp rechts davon. Die Werte zusammengesetzter Datenstrukturen (wie Arrays und Strukturen) werden in C-Programmen mit geschweiften Klammern (`{ }`) angegeben. Wenn nicht genügend Platz vorhanden ist, um den gesamten Namen oder Ausdruck anzugeben, wird er abgeschnitten.

Wenn Sie einen Ausdruck eingeben, der überwacht werden soll, können Sie ohne weiteres einen Variablennamen angeben, der noch nicht definiert ist, weil etwa die Funktion, in der die Variable deklariert ist, noch nicht aufgerufen wurde. Dadurch können Sie auch einen *Watch* -Ausdruck eingeben, bevor der Gültigkeitsbereich für den Ausdruck aktiv ist. (In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Behandlung des Gültigkeitsbereichs, und wann eine Variable oder ein Parameter definiert ist).

Achtung: Das ist die einzige Situation, in der Sie in TDW einen Ausdruck eingeben können, der nicht sofort ausgewertet wird. Das bedeutet, daß ein Tippfehler in einem Namen nicht entdeckt wird, da TDW davon ausgeht, daß der eingegebene Name für eine Variable steht, die erst im Laufe der Programmausführung verfügbar ist.

Falls Sie den Gültigkeitsbereich nicht, wie in Kapitel 9, »Ausdrücke«, besprochen, ausdrücklich angeben, wertet TDW die Ausdrücke im *Watches* -Fenster nur mit dem Bezugsrahmen der aktuellen Position aus, an der Ihr Programm angehalten wurde. Daher haben die Ausdrücke in einem *Watches* -Fenster denselben Wert, den der Ausdruck in der aktuellen Position in Ihrem Programm hätte. Wenn ein *Watch* -Ausdruck einen Variablennamen enthält, der im aktuellen Bezugsrahmen nicht erreicht werden kann – da die Variable in einem anderen Modul zum Beispiel statisch deklariert wurde –, dann ist der Ausdruck nicht definiert und wird mit vier Fragezeichen (`????`) dargestellt.

Wenn Sie eine eine Elementfunktion einer Klasse Schritt für Schritt durchgehen, können Sie den Parameter `this` dem *Watches* -Fenster hinzufügen.

Das lokale Menü des Watches-Fensters

Wie alle lokalen Menüs erhält man das lokale Menü des *Watches* -Fensters mit Alt F10. Wenn die Strg -Tastenkürzel aktiviert sind, kann man einen Menübefehl direkt mit Strg und dem ersten Buchstaben des Befehls aufrufen.

Watch...
Der Befehl *Watch* (engl. beobachten) verlangt nach einem Variablennamen oder einem Ausdruck, der in die Liste im *Watches* -Fenster eingefügt werden soll. Die Eingabe wird am Ende der Liste angehängt.

Edit...
Mit dem Befehl *Edit* kann man einen Ausdruck im *Watches* -Fenster editieren. Man kann einen vorhandenen *Watch* -Eintrag ändern oder einen neuen eingeben.

Sie können diesen Befehl auch mit der Eingabetaste CR oder *OK* aufrufen, wenn der Auswahlbalken über dem gewünschten *Watches* -Eintrag steht.

Remove
Das gerade ausgewählte Objekt im *Watches* -Fenster kann mit dem Befehl *Remove* (engl. entfernen) aus der *Watches* -Liste entfernt werden.

Delete all
Wenn Sie alle Einträge im *Watches* -Fenster löschen wollen, geben Sie den Befehl *Delete all* (engl. alle löschen) ein. Verwenden Sie den Befehl *Watch* , um weitere Variablen zu beobachten. Dieser Befehl wird benötigt, wenn man von einem Programmbereich in einen anderen Bereich springt, in dem die angegebenen Variablen nicht mehr von Bedeutung sind. (Anschließend können Sie mit *Watch* die anderen Variablen neu eingeben.)

Inspect
Mit dem Befehl *Inspect* (engl. untersuchen) wird ein *Inspector*-Fenster geöffnet, in dem der Inhalt des hervorgehobenen Objekts untersucht werden kann. Wenn das Objekt eine zusammengesetzte Datenstruktur hat (Array, Record oder Struktur), wird damit die Untersuchung aller Elemente ermöglicht und nicht nur derjenigen Elemente, die in das *Watches* -Fenster passen. (Der nachfolgende Abschnitt »Inspector-Fenster« enthält alle Informationen über diese Möglichkeiten.)

Change
Der Befehl *Change* (engl. verändern) erlaubt die Veränderung des gerade hervorgehobenen Objekts im *Watches* -Fenster. Im Dialogfenster können Sie den neuen Wert eingeben. Wenn die aktuell eingestellte Sprache es erlaubt, führt TDW alle notwendigen Typumwandlungen genauso durch, als ob der passende Zuweisungsoperator verwendet worden wäre. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über den Zuweisungsoperator und die Datentypumwandlung.

Inspector-Fenster

In einem *Inspector*-Fenster werden Ihre Programmdaten entsprechend dem jeweiligen Datentyp dargestellt. Die *Inspector*-Fenster verhalten sich bei Skalaren (zum Beispiel `char` oder `int`), Zeigern (`char *`), Arrays (`long x[4]`), Funktionen und Strukturen unterschiedlich.

Das *Inspector*-Fenster führt alle Objekte auf, aus denen sich die Datenstruktur zusammensetzt. Die Überschrift enthält den Datentyp der untersuchten Daten und den Namen, falls einer angegeben wurde.

Der erste Eintrag in einem *Inspector*-Fenster ist immer die Speicheradresse des untersuchten Datenobjekts, die als Segment:Offset-Paar angegeben wird, sofern es sich nicht um eine Registervariable oder eine Konstante handelt.

Um den Inhalt eines *Inspector*-Fensters in Form von Bytes zu untersuchen, muß man den Befehl *View/Dump* angeben, während man sich im *Inspector*-Fenster befindet. Das *Dump*-Fenster erscheint dann mit dem entsprechenden Ausschnitt des *Inspector*-Fensters. Man kann zum *Inspector*-Fenster zurückkehren, indem man das Fenster mit *Window/Close* , Alt F3 oder durch Anklicken des Schließsymbols mit der Maus schließt.

In folgenden Abschnitten beschreiben wir die verschiedenen *Inspector*-Fenster, die in den jeweiligen Sprachen von TDW unterstützt werden: C++ und Assembler. Die verwendete Sprache bestimmt das Format der im *Inspector*-Fenster dargestellten Informationen. Datenobjekte und deren Werte erscheinen in einem Format, das der Deklaration in der Quelldatei sehr ähnlich ist.

Merken Sie sich, daß Sie nichts Spezielles unternehmen müssen, um die verschiedenen *Inspector*-Fenster zu erzeugen. Der rechte Ausschnitt erscheint automatisch so, wie es den untersuchten Daten entspricht.

Inspector-Fenster für C-Daten

Skalare

Die skalaren *Inspector*-Fenster zeigen Ihnen die Werte von einfachen Datenobjekten, wie zum Beispiel:

```
char c = 4 ;
unsigned long y = 123456L ;
```

Diese *Inspector*-Fenster enthalten nur eine einzelne Zeile mit Informationen. Unter der obersten Zeile wird die Adresse der Variablen angegeben. Auf der linken Seite steht der Datentyp des Skalars (`char`, `unsigned long` usw.) und auf der rechten Seite steht der Wert. Der Wert kann dezimal, hexadezimal oder in beiden Formaten angegeben werden. Er wird normalerweise zuerst dezimal angegeben, wobei der Hexadezimalwert in Klammern steht (wobei die Standardvorgabe `0x` für hexadezimale Zahlen in C angegeben wird). Mit TDWINST können Sie die Darstellung der Werte ändern.

Wenn die dargestellte Variable den Datentyp `char` hat, wird zusätzlich das entsprechende Bildschirmzeichen dargestellt. Falls der aktuelle Wert keinem druckbaren Zeichen entspricht, müssen Sie einen Backslash (\) mit dem Hexadezimalwert verwenden. Der Wert des Zeichens wird vor den Dezimal- oder Hexadezimalwerten angegeben.

Abbildung 6.3
Ein Inspector-Fenster für einen Skalar in C

Zeiger

In den *Inspector*-Fenstern für Zeiger werden die Werte der Datenelemente angezeigt, die auf andere Datenelemente zeigen:

```
char *p = "abc" ;
int *ip = 0 ;
int **ipp = &ip ;
```

Normalerweise haben die *Inspector*-Fenster von Zeigern nur eine einzige Informationszeile, in der die Adresse der Variablen steht. Auf der linken Seite erscheint `[0]`, womit angedeutet wird, daß es sich um das erste Element eines Arrays handelt. Auf der rechten Seite erscheint der Wert, der vom Zeiger adressiert wird. Wenn der Wert eine komplexe Datenstruktur angibt, zum Beispiel ein Array oder eine Struktur, wird soviel wie möglich davon dargestellt, wobei die Werte in geschweiften Klammern, { }, stehen.

Wenn der Zeiger auf einen `char`-Typ und vielleicht auf einen mit NUL abgeschlossenen String zeigt, erscheinen weitere Informationen. Es wird der Wert jedes Zeichens im String dargestellt. Auf der linken Seite erscheint der Array-Index (`[1]`, `[2]`, usw.) und der Wert wird rechts angegeben, so als wäre es ein Skalar. In diesem Fall wird der gesamte String mit seiner Adresse auch noch in der Überschrift angegeben.

Sie können auch mehrere Zeilen anzeigen, indem Sie den Befehl *Range* des lokalen Menüs verwenden. Dies ist eine sehr wichtige Methode für C-Programmierer, die mit Hilfe von Zeigern auf Array-Elemente oder einzelne Elemente verweisen. Wenn Sie zum Beispiel folgende Variablen deklariert haben:

```
int Array[10];
int *ArrayZgr = Array;
```

und wissen wollen, was `ArrrayZgr` adressiert, verwenden Sie den Befehl *Range* für *ArrayZgr*, geben als Anfangsindex 0 und als Bereich 10 ein. Ohne diese zusätzlichen Angaben sehen Sie nur die ersten Elemente des Arrays.

Die *Inspector*-Fenster von Zeigern haben außerdem einen unteren Ausschnitt, in dem der Datentyp angegeben wird, auf den der Zeiger verweist.

Strukturen und Varianten

Die *Inspector*-Fenster für Strukturen und Varianten zeigen die Werte der Elemente in einer Struktur oder einer Varianten, zum Beispiel:

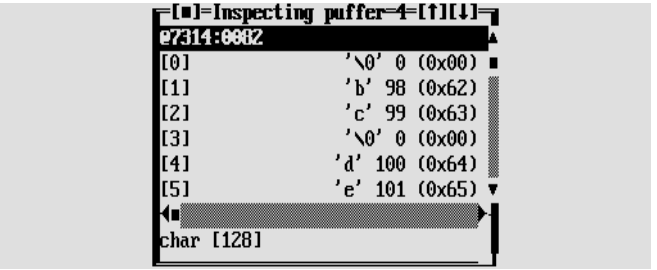


Abbildung 6.4 Ein Inspector-Fenster für einen Zeiger in C

```
struct linfo {
  unsigned int count;
  unsigned int firstletter;
} letterinfo [26];

union {
  int small ;
  long large;
} holder;
```

Diese *Inspector*-Fenster haben einen zweiten Ausschnitt unterhalb des ersten, in dem die Werte der Elemente angegeben sind. Dieser zusätzliche Ausschnitt zeigt den Datentyp des hervorgehobenen Elements im oberen Ausschnitt an.

Abbildung 6.5 Ein Inspector-Fenster für eine Struktur oder Variante in C

Strukturen und Varianten sehen in *Inspector*-Fenstern gleich aus. Dem unteren Ausschnitt des *Inspector*-Fensters kann man entnehmen, ob es sich um eine Struktur oder um eine Variante handelt. Diese *Inspector*-Fenster enthalten nach der Adresse genau so viele Datenobjekte, wie Elemente in der Struktur oder Variante vorhanden sind. Der Name des Elements wird auf der linken Seite angezeigt, der aktuelle Wert dieses Elements auf der rechten Seite, wobei das Format verwendet wird, das dem C-Datentyp entspricht.

Arrays

Inspector-Fenster für Arrays zeigen die Werte der Datenobjekte eines Arrays an, zum Beispiel:

```
long thread [3][4][5] ;
char message[] = "friß oder stirb" ;
```

Jedes Element des Arrays nimmt eine Zeile ein. Auf der linken Seite wird der Array-Index des Elements angegeben. Auf der rechten Seite erscheint der Wert

des Elements. Wenn der Wert ein komplexes Datenobjekt ist, wie eine Struktur oder ein Array, dann wird soviel wie möglich davon dargestellt.

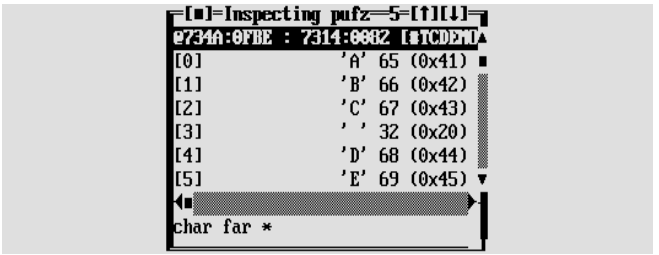


Abbildung 6.6 Ein Inspector-Fenster für ein Array

Sie können auch den Befehl *Range* des lokalen Menüs verwenden, um einen beliebigen Ausschnitt des Arrays zu untersuchen. Dies empfiehlt sich vor allem, wenn das Array aus vielen Elementen besteht und man einen Ausschnitt aus der Mitte des Arrays kontrollieren will. Wenn Sie *Range* wählen, werden Sie zur Eingabe eines Start-Index, eines Kommas und der Anzahl der zu untersuchenden Elemente aufgefordert

Funktionen

Das *Inspector*-Fenster einer Funktion gibt den Funktionstyp in der Überschrift an. Jeder Parameter, mit dem eine Funktion aufgerufen wurde, erscheint hinter der Speicheradresse am Anfang der Liste.

Abbildung 6.7 Ein Inspector-Fenster für eine Funktion in C

Außerdem erhalten Sie Informationen über die Aufrufparameter und den Typ des Rückgabewerts sowie die Aufrufkonventionen. Der untere Ausschnitt gibt den Datentyp des Rückgabewerts der Funktion an.

Inspector-Fenster für Assemblerdaten

Skalare

Die *Inspector*-Fenster für Skalare in Assembler zeigen Ihnen die Werte von einfachen Datenobjekten, wie zum Beispiel:

```
Var1    DW  99
MAGIE   DT  4.608
GROSS   DD  123456
```

Diese *Inspector*-Fenster enthalten nur eine einzelne Zeile an Informationen, die der obersten Zeile mit der Adresse der Variablen folgt. Auf der linken Seite steht der Datentyp des Skalars (`byte`, `word`, `dword`, `qword` usw.), auf der rechten Seite steht der Wert. Dieser Wert kann dezimal, hexadezimal oder in beiden Formaten angegeben werden. Er wird normalerweise dezimal dargestellt, gefolgt von dem in Klammern stehenden Hexadezimalwert (mit dem Turbo-Assembler-Suffix `h`). Mit TWDINST können Sie die vorgegebene Darstellung der Werte ändern.

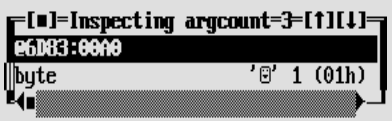


Abbildung 6.8 Skalare Assembler-Daten in einem Inspector-Fenster

Zeiger

Das *Inspector*-Fenster für Zeiger in Assembler enthält die Werte der Datenobjekte, die auf andere Datenobjekte zeigen, zum Beispiel:

```
X       DW   0
XPTR    DW   X
FARPTR  DD   X
```

Normalerweise enthalten *Inspector*-Fenster für Zeiger nur eine einzige Informationszeile, in der die Adresse der Variablen steht.

Auf der linken Seite steht `[0]`, womit angedeutet wird, daß es sich um das erste Element eines Array handelt. Auf der rechten Seite erscheint der Wert, der von dem Zeiger adressiert wird. Wenn der Wert eine komplexe Datenstruktur angibt, zum Beispiel ein Array oder eine Struktur, wird soviel wie möglich davon dargestellt, wobei die Werte in geschweiften Klammern, { }, stehen.

Wenn der Zeiger als Zeiger auf einen `byte`-Datentyp deklariert ist und auf einen mit NUL abgeschlossenen String zeigt, werden die Werte aller Zeichen dieses Strings (entspricht einem Zeichen-Array) dargestellt. Auf der linken Seite des Fensters erscheint der Array-Index (`[1]`, `[2]` usw.), der jeweilige Wert wird rechts ausgegeben, so als sei jedes Array-Elemant ein Skalar. In diesem Fall wird der gesamte String mit seiner Adresse auch noch in der Überschrift angezeigt.

Sie können auch mehrere Zeilen darstellen, wenn Sie das *Inspector*-Fenster öffnen, den Befehl *Range* des lokalen Menüs wählen und eine Zahl größer 1 eingeben.

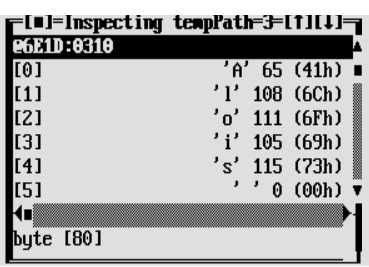


Abbildung 6.9 Ein Inspector-Fenster mit einem Zeiger in einem Assembler-Programm

Arrays

In einem *Inspector*-Fenster für Assembler-Arrays werden die Werte der Datenobjekte eines Array angezeigt, zum Beispiel:

```
WARRAY      DW  10 DUP (0)
MELDUNG     DB  "Gruesse"
```

Jedes Array-Element erscheint in einer eigenen Zeile. Auf der linken Seite des Fensters wird der Index des Array-Elements angegeben; auf der rechten Seite erscheint der adressierte Wert. Wenn der Wert ein komplexes Datenobjekt enthält, wie zum Beispiel eine Struktur `struc`, werden so viele Werte wie möglich dargestellt.

Sie können auch den Befehl *Range* des lokalen Menüs verwenden, um einen beliebigen Ausschnitt des Array zu untersuchen. Dies empfiehlt sich vor allem, wenn das Array aus vielen Elementen besteht, Sie jedoch einen Ausschnitt aus der Mitte des Array kontrollieren will.

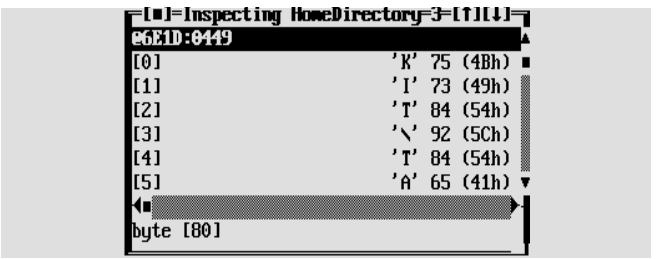


Abbildung 6.10 Ein Inspector-Fenster mit einem Array in Assembler

Strukturen und Varianten

Die *Inspector*-Fenster für Strukturen und Varianten in Assembler zeigen die Werte der Elemente in Strukturen oder Varianten an, zum Beispiel von:

```
X           STRUC
Mem1        DB   ?
Mem2        DD   ?
X           ENDS
ANX         X    <1,ANX>
Y           UNION
ASBYTES     DB  10 DUP (?)
ASFLT       DT  ?
Y           ENDS
AY          Y    <?,1.0>
```

Diese *Inspector*-Fenster haben einen zweiten Ausschnitt, in dem die Werte der Elemente ausgegeben werden. Dieser zusätzliche Ausschnitt zeigt den Datentyp des aktuellen Elements im oberen Ausschnitt an.

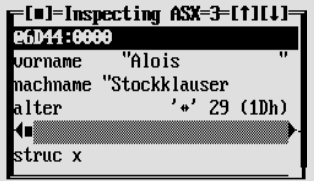


Abbildung 6.11 Ein Inspector-Fenster mit einer Struktur in Assembler

Das lokale Menü des Inspector-Fensters

Die Befehle in diesem Menü machen die Leistungsfähigkeit des *Inspector*-Fensters aus. Wenn Sie den Befehl *Inspect* im lokalen Menü auswählen, können Sie weitere *Inspector*-Fenster anlegen, mit denen Sie Ihre Datenstrukturen kontrollieren können. Mit anderen Befehlen in diesem Menü kann man den Bereich der dargestellten Werte angeben und eine neue Variable untersuchen.

Um das lokale Menü des *Inspector*-Fensters zu erhalten, drücken Sie Alt F10 . Wenn die Strg -Tastenkürzel aktiviert sind, können Sie die Befehle direkt mit Strg und dem ersten Buchstaben des Befehls aufrufen.

Range...

Mit dem Befehl *Range* (engl. Bereich) legen Sie das erste Element und die Anzahl der Elemente fest, die dargestellt werden sollen. Verwenden Sie diesen Befehl, wenn Sie ein Array untersuchen und nur einen bestimmten Unterbereich der Elemente des Arrays sehen wollen. Wenn Sie ein sehr großes Array haben und nur auf einige wenige Elemente in der Mitte zugreifen wollen, können Sie diesen Befehl dazu verwenden, um das *Inspector*-Fenster bei dem Array-Index zu öffnen, den Sie untersuchen wollen. Dieser Befehl ist besonders in C sehr nützlich, wo man sehr oft einen Zeiger auf ein Datenobjekt deklariert, zum Beispiel `char *p`, der aber meistens nicht als Zeiger auf ein einzelnes Zeichen, sondern als Zeiger auf ein Array von Zeichen gedacht ist.

Change...

Der Befehl *Change* (engl. verändern) setzt den Wert des gerade hervorgehobenen Datenobjekts auf den Wert, welchen Sie im Dialogfenster eingeben. Wenn es die aktuelle Sprache erlaubt, führt TDW alle notwendigen Typumwandlungen genauso durch wie der entsprechende Zuweisungsoperator. In Kapitel 9, »Ausdrücke«, erhalten Sie weitere Informationen über den Zuweisungsoperator und Typumwandlungen.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein neues *Inspector*-Fenster, in dem der Inhalt des gerade selektierten Datenobjekts dargestellt wird. Das ist sehr nützlich, wenn im *Inspector*-Fenster selbst weitere Datenobjekte enthalten sind (wie bei Strukturen eines Arrays) und Sie alle diese Objekte sehen wollen.

Wenn das aktuelle *Inspector*-Fenster eine Funktion untersucht, dann bekommen Sie mit dem Befehl *Inspect* den Quellcode der Funktion angezeigt.

Wenn der Auswahlbalken auf dem gewünschten Datenobjekt steht, können Sie diesen Befehl auch mit der Taste CR aktivieren. Mit Esc, können Sie zum vorherigen *Inspector*-Fenster zurückkehren und das *Inspector*-Fenster wieder schließen. Wenn Sie eine Datenstruktur vollständig untersucht haben und alle *Inspector*-Fenster auf einmal entfernen wollen, können Sie den Befehl *Window/Close* oder Alt F3 eingeben. Mit der Maus klicken Sie dazu auf das Schließsymbol.

Descend

Der Befehl *Descend* (engl. herabsteigen) funktioniert wie der Befehl *Inspect*, mit dem Unterschied, daß ein neues Datenobjekt im aktuellen *Inspector-Fenster* eingefügt wird, statt ein neues *Inspector*-Fenster für das Datenobjekt zu öffnen. Der Befehl ist eine Kombination von *New expression* und *Inspect*.

Achtung: Wenn Sie sich einmal in eine Datenstruktur begeben haben, dann können Sie zur alten Datenstruktur nicht mehr zurück. Verwenden Sie diesen Befehl, wenn Sie sich durch eine komplizierte Datenstruktur arbeiten oder eine verkettete Liste durchsuchen wollen, aber keinen Wert darauf legen, zur vorherigen Ebene zurückzukehren. Dadurch kann man die Anzahl der *Inspector*-Fenster auf dem Bildschirm sinnvoll einschränken.

New expression...

Beim Befehl *New expression* (engl. neuer Ausdruck) werden Sie um einen Variablennamen oder einen Ausdruck gebeten, der untersucht werden soll, ohne daß Sie ein weiteres *Inspector*-Fenster anlegen müssen. Damit kann man andere Daten untersuchen, ohne mehrere *Inspector*-Fenster auf dem Bildschirm halten zu müssen. Verwenden Sie diesen Befehl, wenn Sie nicht mehr an den Daten im aktuellen *Inspector*-Fenster interessiert sind. Die *Inspector*-Fenster für Klassen in Turbo C++for Windows unterscheiden sich etwas von den normalen *Inspector*-Fenstern. In Kapitel 10, »Objektorientiertes Debuggen«, finden Sie eine Beschreibung der *Inspector*-Fenster für Objekttypen und Klassen.

Type Cast...

Damit können Sie einen anderen Datentyp (`char*`, `int`, `gh2fp`, `lh2pf`) für das ausgewählte Datenelement vorgeben. Dies empfiehlt sich vor allem bei Symbolen, für die keine Datentypinformationen vorliegen, oder typenlose Zeiger.

In Kapitel 11, »Debuggen unter Windows«, erhalten Sie Informationen über die Verwendungsweise von `gh2fp` und `lh2fp`.

7. Breakpoints

TDW verwendet den Begriff »Breakpoint« (engl. Abbruchs- oder Haltepunkt), um die drei Funktionen zu beschreiben, die bei anderen Debuggern als Breakpoints, Watchpoints und Tracepoints bezeichnet werden.

Die drei Begriffe Breakpoint, Watchpoint und Tracepoint werden traditionell wie folgt definiert: Ein *Breakpoint* ist eine Stelle in Ihrem Programm, an der Sie die Programmausführung anhalten wollen, damit Sie sich Programmvariablen und Datenstrukturen genauer anschauen können. Ein *Watchpoint* dient dazu, daß Ihr Programm angehalten wird, wenn der Wert eines Ausdrucks TRUE wird. Ein *Tracepoint* bewirkt, daß Ihr Programm Zeile für Zeile ausgeführt wird, wenn sich der Wert einer bestimmten Programmvariablen oder ein Ausdruck, der sich auf eine Speicherstelle bezieht, ändert.

TDW vereint diese drei Konzepte, indem ein Breakpoint als dreiteilige Struktur aufgefaßt wird, die aus den folgenden drei Teilen besteht:

- der *Ort*, an dem der Breakpoint gesetzt wird;
- die *Bedingung*, die für einen Breakpoint zutreffen muß;
- die *Aktion*, die beim Auslösen eines Breakpoint ausgeführt werden soll.

Der *Ort* kann entweder eine bestimmte Stelle in Ihrem Programm oder global vorgegeben sein, wobei der Breakpoint dann an jeder Stelle in Ihrem Programm auftreten kann.

Die *Bedingung* kann zutreffen

- immer;
- wenn ein Ausdruck gültig wird;
- wenn sich ein Datenobjekt ändert;
- wenn eine Windows-Botschaft eintrifft.

Es kann auch vorgegeben werden, wie oft eine Bedingung zutreffen muß, bevor die Ausführung angehalten wird.

Bei der *Aktion* gibt es die folgenden Möglichkeiten:

- Programmausführung unterbrechen (ein Breakpoint im engeren Sinne);
- Werte von Variablen aufzeichnen;

- Ausdrücke ausführen (Code einfügen);
- eine Gruppe von Breakpoints aktivieren;
- eine Gruppe von Breakpoints deaktivieren.

In diesem Kapitel zeigen wir Ihnen, wie man mit den Breakpoints in TDW wesentlich effektiver arbeiten kann als mit traditionellen Breakpoints, Watchpoints und Tracepoints. Sie werden das *Breakpoints*-Fenster und das *Log*-Fenster kennenlernen, werden sehen, wie man einfache Breakpoints einsetzt, Breakpoints mit Bedingungen formuliert und Breakpoints zum Protokollieren von Variablen verwendet. Schließlich werden wir zeigen, wie man einen Breakpoint setzt, um den genauen Zeitpunkt zu erfassen, an dem eine bestimmte Variable, ein Ausdruck oder ein Datenobjekt seinen Wert ändert.

Häufig werden Sie nur einige einfache Breakpoints setzen, damit Ihr Programm die Ausführung unterbricht, wenn eine bestimmte Programmzeile ausgeführt wurde. Sie können einen Breakpoint an jeder Stelle Ihres Programms setzen, indem Sie den Cursor auf die entsprechende Zeile positionieren und die Funktionstaste F2 drücken. In gleicher Weise läßt sich ein Breakpoint auch im Maschinencode setzen, indem man im *Code*-Ausschnitt des *CPU*-Fensters die Taste F2 drückt.

Wenn Sie eine Maus haben, klicken Sie einfach die ersten beiden Spalten der Zeile an, in der Sie den Breakpoint setzen wollen. (Wenn Sie die korrekte Spalte angeklickt haben, erscheint ein Stern als Positionsanzeige.)

Es gibt zwei Möglichkeiten, um auf Dialogfenster zuzugreifen, in denen Breakpoints gesetzt oder angepaßt werden können. Das Menü *Breakpoints* bietet eine schnelle Möglichkeit zum Setzen von Breakpoints. Das *Breakpoints*-Fenster liefert eine Übersicht aller gesetzten Breakpoints und einen Zugriff auf Dialogfenster, in denen die Breakpoint-Einstellungen vorgenommen werden.

Das Menü Breakpoints

Das globale *Breakpoints*-Menü können Sie jederzeit über die Tastenkombination Alt B aufrufen.

Toggle

Mit dem Menüpunkt *Toggle* (engl. umschalten) kann in der aktuelle Zeile im *Module*-Fenster oder im Code-Ausschnitt des *CPU*-Fensters einen Breakpoint setzen oder, falls dort bereits ein Breakpoint gesetzt ist, diesen löschen. Die Taste F2 dient dafür als Tastenkürzel.

At...

Unter dem Eintrag *At...* (engl. an der Stelle...) kann eine bestimmte Stelle in Ihrem Programm als Breakpoint ausgewählt werden. In dem Fenster, das sich anschließend öffnet, können Sie die Optionen für diesen Breakpoint angeben. Das Tastenkürzel lautet Alt F2 . (Weiter hinten in diesem Kapitel finden Sie eine ausführliche Beschreibung zum Dialogfenster *Breakpoints Options* .)

Changed memory global

Mit dem Befehl *Changed memory global...* (engl. Speicherveränderung global) wird ein globaler Breakpoint gesetzt, der zur Unterbrechung der Programmausführung führt, wenn ein bestimmter Speicherbereich verändert wird. Geben Sie im dazugehörigen Eingabefeld *Enter Memory Address/Count* den Speicherbereich an, der überwacht werden soll. TDW prüft den hier eingegebene Variablenausdruck jedesmal, wenn eine Quelltextzeile ausgeführt wurde. Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Das lokale Menü des Breakpoints-Fensters« beim Befehl *Changed memory* .

Expression true global...

Der Menüpunkt *Expression true global* (engl. Ausdruck zutreffend global) dient zur Definition globaler Breakpoints, die dann ausgelöst werden, wenn ein Ausdruck global gültig (TRUE) wird. Geben Sie im Eingabefeld *Enter Expression for Condition Breakpoint* den Ausdruck ein, dessen globale Gültigkeit geprüft werden soll. TDW prüft den hier eingegebenen Ausdruck jedesmal, wenn eine Quelltextzeile ausgeführt wurde. Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Bedingte Breakpoints«.

Hardware breakpoint...

Mit Hilfe dieses Befehls können Sie auf das Dialogfenster *Hardware Breakpoints Options* zugreifen. Für Hardware-Debuggen muß Ihr Computer-System entsprechend eingerichtet sein. Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Hardware-Breakpoints«.

Delete all

Der Befehl *Delete All* (engl. alle löschen) entfernt alle gesetzten Breakpoints.

Das Fenster Breakpoints

Das *Breakpoints* -Fenster wird durch die Auswahl des Befehls *View/Breakpoints* geöffnet. Darin können Sie sich die einzelnen Breakpoints im Überblick ansehen und die einzelnen Bedingungen anpassen, bei deren Eintreten ein Breakpoint ausgelöst wird. In diesem Fenster können Sie neue Breakpoints hinzufügen und bestehende Breakpoints löschen oder verändern.

Breakpoints -Fenster sind in zwei Ausschnitte unterteilt. Der linke Ausschnitt mit der Breakpoint-Liste zeigt alle Adressen, an denen Breakpoints gesetzt wurden. Der rechte Ausschnitt mit den Breakpoint-Details enthält nähere Informationen über den aktuellen Breakpoint, der im linken Feld durch den Auswahlbalken markiert ist.

Nur für den linken Ausschnitt gibt es ein lokales Menü, das über Alt F10 aufgerufen wird. Falls die Srtg -Tastenkürzel eingeschaltet sind, können Sie den gewünschten Befehl mit Strg und dem ersten Buchstaben des Menübefehls direkt ausführen.

Abbildung 7.1
Das Fenster Breakpoints

Das lokale Menü des Breakpoint-Fensters

Dieses Menü bietet Befehle für das Hinzufügen neuer Breakpoints und zum Löschen oder zum Verändern der Arbeitsweise bestehender Breakpoints.

Set options...

Der Befehl *Set Options* öffnet das Dialogfenster *Breakpoint options* , in welchem Sie folgende Einstellungen vornehmen können:

- einen Breakpoint global vorgeben;
- einen Breakpoint ein- oder ausschalten;
- einen Breakpoint in eine Gruppe einfügen;
- auf das Dialogfenster *Conditions and Actions* zugreifen.

Zusätzliche Informationen über das Dialogfenster *Breakpoint Options* erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Das Dialogfenster Breakpoint Options«.

Add...

Durch den Befehl *Add* (engl. hinzufügen) wird ein Dialogfenster ähnlich *Set options* geöffnet. Der Unterschied liegt darin, daß der Cursor in einem leeren Textfeld *Address* steht. Im Eingabefeld von *Address* müssen Sie eine Adresse einfügen. Wenn Sie zum Beispiel einen Breakpoint in Zeile 3201 Ihres C-Quellcodes setzen wollen, geben Sie #3201 ein. Falls sich die fragliche Zeile in einem Modul befindet, das nicht im *Module*-Fenster angezeigt wird, geben Sie das Nummerzeichen (#), den Namen des Moduls, ein weiteres Nummernzeichen und die Zeilennummer ein. Zum Beispiel: `#OTHERMOD#3201`.

Sie können einen Breakpoint auch ganz einfach einfügen, indem Sie die Adresse eingeben, an der der Breakpoint gesetzt werden soll. Es erscheint dann ebenfalls ein Dialogfenster, so als ob Sie den *Add*-Befehl ausgewählt hätten.

Sobald der Breakpoint eingegeben wurde, können Sie die anderen lokalen Menübefehle verwenden, um die Arbeitsweise des Breakpoints zu verändern.

Remove

Der Befehl *Remove* (engl. entfernen) entfernt den gerade hervorgehobenen Breakpoint.

Delete all

Der Befehl *Delete all* (engl. alle löschen) entfernt alle Breakpoints, sowohl die globalen als auch diejenigen, die einer bestimmten Adresse zugeordnet sind. Sie müssen anschließend wieder weitere Breakpoints angeben, wenn Sie wollen, daß Ihr Programm an einem Breakpoint anhalten soll. Verwenden Sie diesen Befehl mit Vorsicht.

Inspect

Der Befehl *Inspect* (engl. untersuchen) zeigt Ihnen die Zeile im Quelltext bzw. die entsprechende Zeile eines Assembler-Befehls, der ein hervorgehobener Breakpoint zugeordnet ist. Wenn der Breakpoint an einer Adresse gesetzt ist, die einer Zeile in Ihrem Quelltext entspricht, wird ein *Module*-Fenster geöffnet, und der Cursor steht auf der Anweisung, für die der Breakpoint gesetzt ist. Ansonsten wird beim Auslösen des Breakpoint das *CPU*-Fenster geöffnet und der entsprechende Befehl im *Code*-Ausschnitt hervorgehoben.

Sie können diesen Befehl auch dadurch aufrufen, indem Sie die Taste CR einmal drücken, wenn der Breakpoint durch den Auswahlbalken in der Breakpoint-Liste hervorgehoben ist.

Group

Mit Hilfe des Befehls *Group* können Sie Breakpoints zu Gruppen zusammenfassen. Einer Breakpoint-Gruppe wird automatisch eine positive ganze Zahl zugeordnet, anhand der Sie die Gruppe identifizieren können. TDW verwendet automatisch eine neue Gruppennummer für jeden Breakpoint, der angelegt wird. Als Gruppennummer wird immer die niedrigste Nummer verwendet, die noch nicht vergeben ist. Falls zum Beispiel die Nummern 1, 2 und 5 bereits von Gruppen belegt sind, wird für den nächsten Breakpoint automatisch die Nummer 3 vergeben.

Die Gruppennummer eines Breakpoint können Sie im Dialogfenster *Break-*

point Options verändern, indem Sie den Breakpoint zu einer Gruppe mit anderen Breakpoints zusammenfassen. Breakpoints, die in Gruppen zusammengefaßt sind, können mit einem einzigen Befehl aktiviert, deaktiviert unt gelöscht werden. Wird der Befehl *Group* über das lokale Menü des *Breakpoints*-Fensters ausgewählt, erscheint das Dialogfenster *Edit Breakpoint Groups* . Dieses Dialogfenster zeigt eine Auflistung aller vorhandenen Breakpoint-Gruppen und ermöglicht Ihnen die Zusammenfassung aller Funktionen innerhalb eines Moduls zu einer Gruppe.

Groups

In der *Groups*-Liste werden die aktuellen Breakpoint-Gruppen angezeigt.

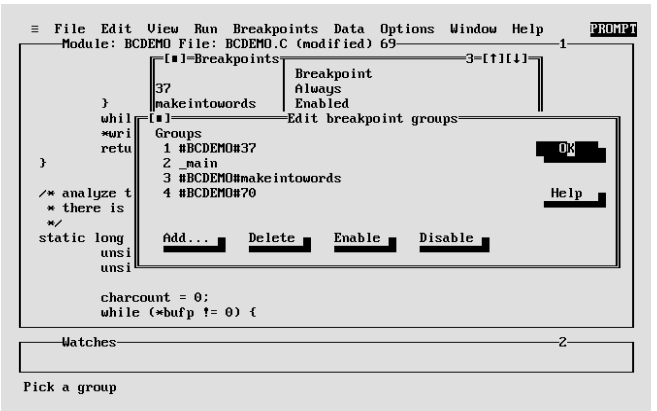


Abbildung 7.2 Das Dialogfenster Edit Breakpoint-Group

Add

Der Schalter *Add* aktiviert das Dialogfenster *Add Group*. Das Dialogfenster *Add Group* enthält eine Liste und zwei Schaltfelder, mit denen Sie alle Funktionen eines einzelnen Moduls oder alle Elementfunktionen einer Klasse in einer Breakpoint-Gruppe zusammenfassen können:

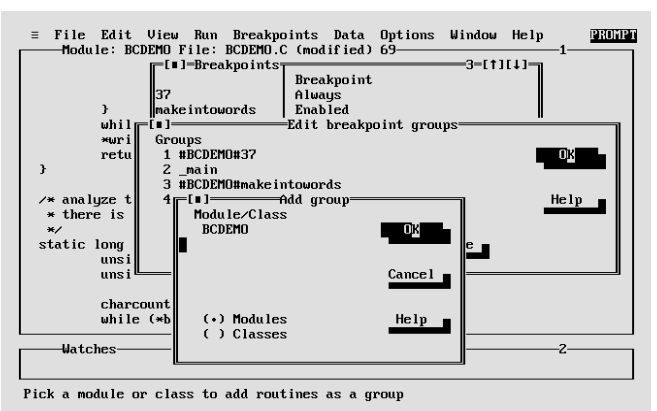


Abbildung 7.3 Das Dialogfenster Add Group

■ In der Liste *Module/Class* wird eine Liste der Module und Klassen angezeigt, die in dem Programm enthalten sind, das im *Module*-Fenster geladen ist. Heben Sie das gewünschte Modul oder die entsprechende Klasse hervor, drücken Sie anschließend *OK*, um für alle Funktionen in einem Modul oder in einer Klasse Breakpoints zu setzen. All diese Breakpoints werden in einer Breakpoint-Gruppe zusammengefaßt.

■ Mit den beiden Schaltfeldern können Sie den Typ der Funktionen auswählen, die in der Liste *Module/Class* angezeigt werden:

∩ Das Schaltfeld *Modules* wählt alle Module des aktuell geladenen Programms aus und zeigt sie in der Liste *Module/Class* an.

∩ Das Schaltfeld *Classes* wählt alle C++ Klassen des aktuellen Programms für die Anzeige in der Liste *Module/Class* aus.

Delete

Der Schalter *Delete* im Dialogfenster *Edit Breakpoint* löscht die gerade hervorgeobene Gruppe aus der Liste *Groups*. Alle Breakpoints dieser Gruppe werden zusammen mit ihren Einstellungen entfernt.

Enable

Der Schalter *Enable* aktiviert eine Breakpoint-Gruppe, die zuvor mit *Disable* ausgeschaltet wurde.

Disable

Der Befehl *Disable* schaltet die in der Liste *Groups* gerade hervorgehobene Breakpoint-Gruppe vorübergehend aus. Ausgeschaltete Breakpoints werden nicht gelöscht; sie werden lediglich in der aktuellen Debugger-Sitzung nicht berücksichtigt. Wird eine Gruppe wieder eingeschaltet, so werden alle Einstellungen der Breakpoints in dieser Gruppe wieder aktiv.

Das Dialogfenster Breakpoint Options

Das Dialogfenster *Breakpoint Options* erreichen Sie über den Befehl *Breakpoints/At* und die Befehle *Set Options* und *Add* des lokalen Menüs im *Breakpoints*-Fenster.

Address

Das Eingabefeld *Address* zeigt die Adreßangabe des aktuellen Breakpoint. In der Regel werden Sie dieses Feld nicht ändern. Falls Sie jedoch den Namen der Adresse ändern wollen, geben Sie den neuen Namen in das Adreßfeld ein.

Group ID

Im Eingabefeld *Group ID* können Sie den aktuellen Breakpoint in eine neue oder bereits vorhandene Gruppe einfügen. Eine Breakpoint-Gruppe ist durch eine positive ganze Zahl bestimmt. Im vorherigen Abschnitt finden Sie weitere Informationen über Breakpoint-Gruppen.

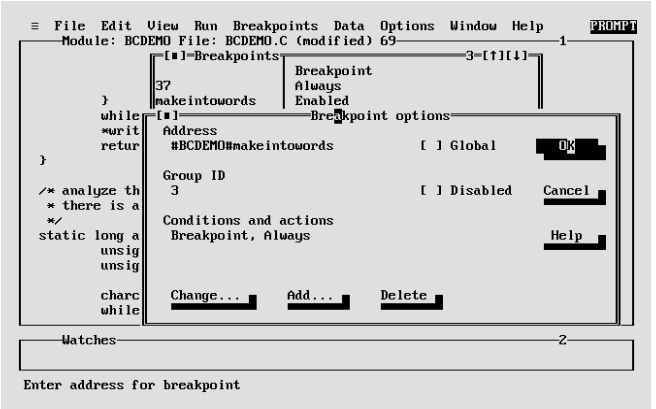


Abbildung 7.4 Das Dialogfenster Breakpoint Options

Global

Das Markierungsfeld *Global* aktiviert, sofern es angekreuzt ist, eine globale Überprüfung. Das bedeutet, daß die Breakpoint-Bedingungen jedesmal überprüft werden, wenn eine Zeile des Quellcodes ausgeführt wird. Da globale Breakpoints nach jeder Ausführung einer Zeile überprüft werden, wird das Feld *Address* mit »not available« belegt, da diese Zuordnung nicht mehr gültig ist.

Beim Setzen eines globalen Breakpoints *m201ss* Sie eine Bedingung vorgeben, unter der der globale Breakpoint ausgelöst wird. Ansonsten ergibt sich ein Breakpoint, der bei jeder Zeile des Quellcodes ausgelöst wird (falls dies gewünscht ist, empfiehlt sich der Befehl *Run/Trace*). Zusätzliche Informationen erhalten Sie im weiteren Verlauf dieses Kapitels im Abschnitt »Globale Breakpoints«.

Disabled

Das Markierungsfeld *Disabled* schaltet den aktuellen Breakpoint aus. Obwohl dieser Befehl so ähnlich funktioniert wie der Befehl *Toggle* aus dem Menü *Breakpoints*, werden durch *Disabled* die Einstellungen eines Breakpoints nicht gelöscht (wie dies beim Befehl *Toggle* der Fall ist). Durch *Disabled* wird der Breakpoint lediglich solange ausgeschaltet, bis er durch eine Aufhebung der Markierung in diesem Feld wieder eingeschaltet wird. Wird der Breakpoint wieder aktiviert, so werden auch alle vorgenommenen Einstellungen des Breakpoints wieder aktiv.

Dieses Markierungsfeld ist sehr nützlich, um einen komplizierten Breakpoint, den Sie momentan nicht wünschen, aber eventuell später noch einmal benötigen, vorübergehend auszuschalten. Damit sparen Sie sich die erneute Definition der Aktionen und Bedingungen des Breakpoints.

Conditions and Actions

Die Liste unter *Conditions and Actions* zeigt die Bedingungen und Aktionen für den aktuellen Breakpoint.

Change

Mit dem Aktionsschalter *Change* wird das Dialogfenster *Conditions and Actions* angezeigt (siehe nachfolgenden Abschnitt). Mit diesem Befehl können Sie den gerade in der Liste hervorgehobenen Eintrag bearbeiten.

Add

Um eine neue Reihe von Bedingungen und Aktionen für den aktuellen Breakpoint festzulegen, wählen Sie den Schalter *Add*. Wie beim Befehl *Change* wird dadurch das Dialogfenster *Conditions and Actions* geöffnet.

Delete

Der Befehl *Delete* entfernt den aktuell hervorgehobenen Eintrag der Liste *Conditions and Actions* aus der Breakpoint-Definition.

Das Dialogfenster Conditions and Actions

Durch Betätigen der Schalter *Change* oder *Add* im Dialogfenster *Breakpoint Options* wird das Dialogfenster *Conditions and Actions* geöffnet.

Wird ein Breakpoint auf eine Zeile des Quellcodes gesetzt, erhält er die Standardeinstellung *Always Break* , d.h. die Programmausführung wird jedesmal unterbrochen, wenn das Programm auf diesen Breakpoint stößt. Im Dialogfenster *Conditions and Actions* können Sie die speziellen Bedingungen vorgeben, durch die der Breakpoint gegebenfalls ausgelöst werden soll, und unterschiedliche Aktionen vorgeben, die beim Auslösen eines Breakpoint ausgeführt werden sollen.

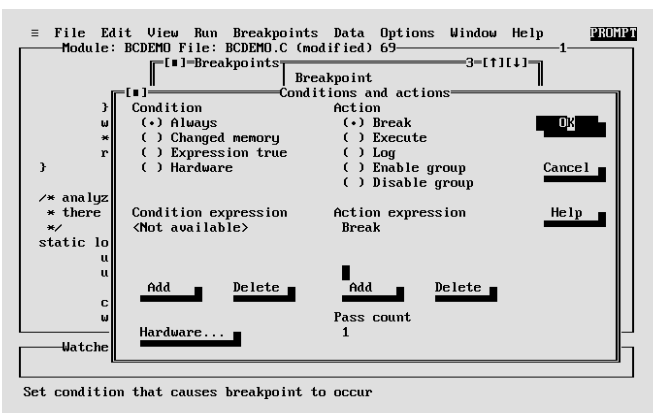


Abbildung 7.5 Das Dialogfenster Conditions and Actions

Die Anpassung der Breakpoints findet mit Hilfe zweier Gruppen von Schaltfeldern und dreier Eingabefelder statt. Außerdem öffnet das Schaltfeld *Hardware* das Dialogfenster *Hardware Breakpoints Options* , in dem Sie die genauen Eigenschaften des Hardware-Breakpoint bestimmen können. Im Abschnitt »Optio-

nen für Breakpoints« im weiteren Verlauf dieses Kapitels erhalten Sie hierzu nähere Informationen.

Die Schaltfelder von Condition

Die Schaltfelder von *Condition* haben vier Einstellungen: Always, Changed Memory, Expression true und Hardware.

Always Bei der Einstellung *Always* (engl. immer) müssen keine zusätzlichen Bedingungen erfüllt werden, wenn der Breakpoint ausgelöst wird; er wird jedesmal ausgelöst, wenn das Programm auf diesen Breakpoint trifft.

Changed Memory Wenn die Option *Changed memory* (engl. Speicherbereich verändert) ausgewählt wurde, wird eine Variable oder ein Objekt im Speicher beobachtet. Ändert sich der betreffende Speicherbereich, wird ein Breakpoint ausgelöst. Geben Sie den Ausdruck für das Objekt, das überwacht werden soll, im Eingabefeld *Condition expression* ein. Weiterführende Informationen erhalten Sie im Abschnitt »Unterbrechungen bei Datenveränderungen« im weiteren Verlauf dieses Kapitels.

Expression true Mit der Einstellung *Expression true* (engl. Ausdruck trifft zu) wird ein Breakpoint dann ausgelöst, wenn ein Ausdruck den Wert TRUE ergibt, also zutrifft. Nach Auswahl dieser Option können Sie den entsprechenden Ausdruck im Eingabefeld *Condition Expression* eingeben. Weitere Informationen zu diesem Thema finden Sie im Abschnitt »Bedingte Breakpoints« weiter hinten in diesem Kapitel.

Hardware Bei der Auswahl der Einstellung *Hardware* wird der Breakpoint über die hardware-gestützten Gerätetreiber ausgelöst. Da Sie die Hardware-Unterstützung nur für globale Breakpoints verwenden können, ist zu beachten, daß das Feld *Global* im Dialogfenster *Breakpoint Options* markiert sein muß, damit Sie auf diese Option zugreifen können.

Sie müssen auf *Hardware* umschalten, bevor der Schalter *Hardware* im unteren Bereich des Dialogfensters aktiviert wird. Wenn Sie diesen Schalter drücken, wird das Dialogfenster *Hardware Breakpoint Options* angezeigt. Die Auswahlmöglichekiten sind in der Datei HDWDEBUG.TD beschrieben. Mit *Hardware/Breakpoint* können Sie auf einfache Weise Hardware-Breakpoints setzen. Weitere Informationen hierzu finden Sie im Abschnitt »HardwareBreakpoints« weiter hinten in diesem Kapitel.

Die Schaltfelder von Action

Mit den Schaltfeldern unter *Actions* können folgende alternativen Einstellung vorgenommen werden:

Break führt zur Unterbrechung der Programmausführung, wenn der Breakpoint ausgelöst wird. Nach dem Anhalten des Programms erscheint der Turbo-Debugger-Bildschirm, und Sie können Befehle eingeben, um die Datenstrukturen in Ihrem Programm zu überprüfen.

Execute Wenn Sie die Option *Execute* (engl. ausführen) für einen Breakpoint vorgeben, wird ein von Ihnen vorgegebener Ausdruck ausgeführt. Nach der Auswahl dieses Menübefehls müssen Sie den entsprechenden Ausdruck eingeben. Der Ausdruck sollte in irgendeiner Form einen Seiteneffekt enthalten, wie zum Beispiel das Setzen einer Variablen auf einen bestimmten Wert. Man kann diese Option auch als »Code-Kitt«, eine Art temporäre Programmergänzung, betrachten, da man einen Ausdruck einfügen kann, der vor dem nachfolgenden Programmteil an der aktuellen Stelle eingefügt wird, so daß man einen Bruch durch einen Programmierfehler notdürftig »kitten« kann.

Natürlich beschränkt sich diese Methode nur auf das Einfügen eines Ausdrucks, bevor eine bereits vorliegende Programmzeile ausgeführt wird; für die Änderung von Programmzeilen ist diese Methode nicht geeignet.

Log Über die Vorgabe *Log* (engl. Protokoll) legen Sie fest, daß der Wert eines Ausdrucks beim Auslösen des Breakpoint im *Log*-Fenster aufgezeichnet werden soll. Nach der Auswahl dieses Menübefehls müssen Sie den Ausdruck eingeben, dessen Wert Sie protokollieren wollen. Achten Sie bitte darauf, daß der Ausdruck keine unerwarteten Seiteneffekte hat. In Kapitel 9, »Ausdrücke«, finden Sie eine Beschreibung von Ausdrücken und möglichen Seiteneffekten.

Enable Group Der Schalter *Enable Group* erlaubt die Aktivierung einer Gruppe von zuvor ausgeschalteten Breakpoints durch das Auslösen eines einzelnen Breakpoints.

Disable Group Der Schalter *Disable Group* erlaubt das Ausschalten einer Gruppe von bislang aktivierten Breakpoints durch das Auslösen eines einzelnen Breakpoints. Beim Ausschalten werden die Breakpoints nicht gelöscht, sondern in der aktuellen Debugger-Sitzung nur zwischenzeitlich deaktiviert.

Bedingungen und Aktionen

Der wichtigste Schritt beim Einstellen von Breakpoints ist das Setzen von Bedingungen, unter denen der Breakpoint ausgelöst werden soll, und die Angabe von Aktionen, die durch den Breakpoint ausgelöst werden sollen. Diese Einstellungen können in zwei Eingeabfeldern spezifiziert werden, nämlich in *Condition Expression* und *Action Expression*.

Condition Expression Falls Sie auf *Changed Memory*, *Expression True* oder *Hardware Condition* umgeschaltet haben, müssen Sie eine Reihe von Bedingungen angeben, damit TDW Informationen darüber hat, wann der Breakpoint ausgelöst werden soll. Die Reihe der Bedingungen können aus einem oder mehreren Ausdrücken bestehen; jede der angegebenen Bedingungen muß zutreffen, damit sie als Gesamtheit zutreffen.

Die Bedingungen können mit mehreren Aktionen verbunden werden. Wenn die Bedingungen zutreffen, werden die entsprechenden Aktionen ausgeführt. Um Bedingungen für einen Breakpoint vorzugeben, gehen Sie wie folgt vor:

1. Wählen Sie entweder den Schalter *Changed Memory*, *Expression True* oder *Hardware*.

2. Drücken Sie den Schalter *Add* unter dem Eingabfeld *Condition Expression*.

3. Tragen Sie die Bedingung oder den Variablenausdruck in das Eingabefeld *Condition Expression* ein.

4. Wenn Sie ein oder mehrere Variablen bzw. Ausdrücke für eine bestimmte Art von Aktion testen wollen, wiederholen Sie Schritt 2 und 3, bis alle Ausdrücke im Eingabefeld *Condition Expression* aufgeführt sind.

5. Wenn alle Bedingungen vollständig sind, tragen Sie im Eingabfeld *Action Expression* die gewünschte(n) Aktion(en) ein, die beim Auslösen des Breakpoints ausgeführt werden soll(en).

Ein einzelner Breakpoint kann mit mehreren Bedingungen und Aktionen verknüpft werden. Falls Sie einem Breakpoint mehr als eine Reihe von Bedingungen und Aktionen zuweisen wollen, wählen Sie *OK* nach der Eingabe der ersten Reihe an Bedingungen und Aktionen. Dadurch wird das Dialogfenster *Conditions and Actions* geschlossen, und Sie kehren wieder zum Dialogfenster *Breakpoint Options* zurück. Dort wählen Sie *Add*, um eine neue Reihe von Bedingungen und Aktionen anzugeben. Weist ein Breakpoint mehrere Reihen von Bedingungen und Aktionen auf, so werden diese nacheinander in der Reihenfolge ausgewertet, in der sie eingegeben wurden. Falls mehr als eine Aktionsreihe zutrifft, werden auch mehrere Aktionen durchgeführt.

Mit dem Schalter *Delete* unterhalb des Eingabefelds von *Condition Expression* können Sie den gerade ausgewählten Ausdruck im Eingabefeld *Condition Expression* löschen. Wählen Sie diesen Schalter aus, wenn Sie eine Bedingung aus der Reihe löschen wollen.

Action Expression Wird das Schaltfeld *Execute*, *Log*, *Enable Group* oder *Disable Group* ausgewählt, muß eine Reihe von Aktionen angegeben werden, damit TDW Informationen darüber hat, was beim Auslösen des Breakpoints zu tun ist. Eine Reihe von Aktionen besteht aus einer oder mehreren Aktionen.

Um einem Breakpoint Aktionen zuzuordnen, gehen Sie wie folgt vor:

1. Wählen Sie den Schalter *Execute*, *Log*, *Enable Group* oder *Disable Group* .

2. Wählen Sie den Schalter *Add* unterhalb des Eingabefelds *Action Expression*.

3. Tragen Sie die gewünschte Aktion in *Action Expression* ein.

4. Um einem Breakpoint mehr als eine Aktion zuzuweisen, wiederholen Sie Schritt 2 und 3, bis alle Aktionen in *Action Expression* aufgeführt sind.

5. Ist die Eingabe der Aktionen abgeschlossen, wählen Sie *OK* im Dialogfenster *Conditions and Actions* .

Wurde einer der Schalter *Enable Group* oder *Disable Group* gewählt, schreiben Sie einfach die Nummer der Breakpointgruppe in das Eingabefeld von *Action Expression*, um die gewünschten Gruppe zu aktivieren oder auszuschalten.

Mit Hilfe des Schalters *Delete* unterhalb des Eingabefelds von *Action Expression* können Sie die gerade ausgewählte Aktion aus der Reihe der Aktionen entfernen.

Pass Count

Im Eingabefeld *Pass count...* (engl. Durchlaufzähler) wird die Anzahl der Durchläufe festgelegt, in denen eine Breakpoint-Bedingung zutreffen muß, bevor der Breakpoint ausgelöst wird. Standardmäßig wird ein Durchlauf vorausgesetzt. Der Durchlaufzähler wird nur dann um 1 vermindert, wenn alle Bedingungen eines Breakpoints erfüllt sind. Das bedeutet, daß der Breakpoint im Falle einer Zusatzbedingung nur dann ausgelöst wird, wenn die Bedingung so oft wie angegeben zutraf.

Optionen für Breakpoints

Außer daß Sie Ihr Programm an einer bestimmten Stelle einfach anhalten, können Sie die Unterbrechungen sehr fein steuern, indem Sie festlegen, wann ein Breakpoint ausgelöst und welche Aktion dann unternommen werden soll.

Einfache Breakpoints

Wenn ein Breakpoint zum ersten Mal gesetzt wird, erhält er zunächst die Standardeinstellung *Always Break* . Für einen einfachen Breakpoint kann man die Aktionen und Bedingungen jederzeit nachträglich ändern. Es gibt eine Reihe von Möglichkeiten, um einen einfachen Breakpoint zu setzen, von denen entsprechend den Umständen die eine oder andere die jeweils bessere Methode darstellt:

- Positionieren Sie den Cursor auf die gewünschte Quellzeile im *Module*-Fenster (oder im Code-Ausschnitt eines *CPU*-Fensters) und führen Sie den Befehl *Breakpoints/Toggle* aus oder drücken Sie die Taste F2 oder klicken Sie die Zeile mit Ihrer Maus an. Wenn Sie diesen Befehl in einer Zeile eingeben, in der bereits ein Breakpoint gesetzt war, wird dieser gelöscht.

- Rufen Sie den lokalen Menübefehl *Add* des Ausschnitts *Breakpoint List* des *Breakpoints* -Fensters auf und geben Sie die Adresse des Programmcodes ein, an dem der Breakpoint gesetzt werden soll. (Eine Adresse im Programmcode hat dasselbe Format wie ein Zeiger der aktuellen Programmiersprache. Siehe Kapitel 9, »Ausdrücke«.)

- Führen Sie den Befehl *Breakpoints/At* aus und geben Sie die Adresse des Programmcodes ein, an dem der Breakpoint gesetzt werden soll.

Globale Breakpoints

Wenn Sie einen Breakpoint global vorgeben, überprüft TDW diesen Breakpoint bei der Ausführung jeder Programmzeile. Wenn die Bedingungen zutreffen, werden die dafür vorgesehenen Aktionen ausgeführt.

Falls Sie eine globale Überprüfung bei jedem Maschinenbefehl wünschen, setzen Sie einen globalen Breakpoint und drücken im CPU-Fenster F9. Diese Art der Programmüberwachung sollten Sie nur dann vornehmen, wenn Sie das Problem auf einen kleinen Programmbereich beschränken können. Im CPU-Fenster können Sie dann die exakte Position ermitteln, an der die Schwierigkeiten auftreten.

Da eine Debugger-Aktion an jeder Zeile des Programm- oder Maschinencodes ausgeführt wird, ergibt sich durch globale Breakpoints eine enorme Verlangsamung im Programmablauf. Seien Sie also vorsichtig mit globalen Breakpoints; sie sollten Sie nur dann verwenden, wenn Sie genau wissen wollen, wann sich eine Variable ändert, eine bestimmte Bedingung zutrifft oder das Programm den Datenbereich zerstört. (Falls Sie Hardware-Breakpoints setzen wollen, müssen Sie *Global* jedoch immer markieren.)

Häufig dienen globale Breakpoints zur Überwachung von Datenänderungen. In diesem Fall überprüft TDW die Änderung von Speicherbereichen nach der Ausführung jeder Programmzeile. Als Alternative für einen globalen Breakpoint könnten Sie auch einen Breakpoint setzen, durch den Änderungen lediglich dann überwacht werden, wenn eine bestimmte Programmzeile ausgeführt wird. Dies ist wesentlich effektiver, da sich dadurch der Aufwand für TDW zum Überprüfen der Änderungen enorm verringert (denn in diesem Fall muß TDW nicht darauf achten, wann sich die Daten ändern, sondern nur darauf, ob sie sich geändert haben). Das Menü *Breakpoints* bietet eine Reihe von Tastenkürzeln für die Definition globaler Breakpoints. Weitere Informationen finden Sie bei der Beschreibung der Optionen *Changed Memory Global* und *Expression True Global* .

Unterbrechung bei Datenveränderungen

Wenn Sie herausfinden wollen, in welchem Teil Ihres Programms ein bestimmter Datenbereich verändert wird, können Sie einen globalen Breakpoint mit einer der zuvor erwähnten Methoden setzen. Anschließend wählen Sie den Schalter *Changed memory* im Dialogfenster *Conditions and Actions* aus und geben einen Ausdruck für den Datenbereich zusammen mit einer optionalen Anzahl an Objekten ein, die überwacht werden sollen. Die Anzahl an Byte im überwachten Bereich ergibt sich aus der Größe des Objekts mal der Anzahl der Objekte. Nehmen wir zum Beispiel an, daß Sie folgendes Array in C deklariert haben:

```
int string[81];
```

Wenn Sie nun eine Änderung in den ersten zehn Elementen dieses Arrays

überwachen wollen, tragen Sie folgenden Ausdruck in das Eingabefeld von *Condisiton Expression* ein:

```
&string[0], 10
```

Der überwachte Bereich ist 20 Byte lang, da ein `int` zwei Byte lang ist und zehn Elemente kontrolliert werden sollen.

Wenn der Breakpoint unter *Changed Memory* global ist, wird Ihr Programm wesentlich langsamer ausgeführt, da der Programmbereich in jeder ausgeführten Programmzeile auf eine Änderung hin überprüft wird.

Wenn Sie einen Hardware-Gerätetreiber installiert haben, versucht TDW einen Hardware-Breakpoint zu setzen, um Veränderungen im Datenbereich aufzuspüren. Die einzelnen Hardware-Debugger unterstützen eine unterschiedliche Anzahl und Art von Breakpoints. Ob ein Breakpoint durch einen Hardware-Breakpoint unterstützt wird, erkennen Sie im *Breakpoints*-Fenster mit Hilfe des Befehls *View/Breakpoints*. Jeder hardware-unterstützte Breakpoint wird in diesem Fenster mit einem Stern (*) gekennzeichnet. Diese Breakpoints sind wesentlich schneller als globale Breakpoints, die nicht hardware-unterstützt sind.

Bedingte Breakpoints

Es gibt viele Gelegenheiten, bei denen ein Breakpoint übersprungen werden soll, wenn eine bestimmte Anweisung ausgeführt wird. Das ist insbesondere dann der Fall, wenn die Zeile im Programmablauf sehr oft ausgeführt werden muß, bevor man überhaupt an dieser Zeile interessiert ist. TDW kennt zwei Möglichkeiten, um festzulegen, wann ein Breakpoint tatsächlich ausgeführt werden soll: *Durchlaufz204hlen* und *Bedingungen*.

Der Gültigkeitsbereich von Breakpoint-Ausdrücken

Sowohl die Aktion, die beim Auslösen eines Breakpoint durchgeführt wird, als auch die Bedingung, unter der der Breakpoint ausgelöst wird, kann durch einen von Ihnen vorgegebenen Ausdruck gesteuert werden. Dieser Ausdruck wird mit Hilfe des Gültigkeitsbereichs der Adresse ausgewertet, an der der Breakpoint gesetzt wurde, und nicht mit dem Gültigkeitsbereich, an dem das Programm angehalten wird. Das bedeutet, daß Ihr Breakpoint-Ausdruck nur die Variablennamen verwenden darf, die an der Adresse in Ihrem Programm gültig sind, an der Sie den Breakpoint gesetzt haben, falls Sie den Gültigkeitsbereich nicht explizit vorgeben. (In Kapitel 9, »Ausdrücke«, finden Sie eine ausführlich Behandlung des Gültigkeitsbereichs und der Vorgaben für einen Gültigkeitsbereich.)

Falls Sie einen Breakpoint für einen Ausdruck in einem Modul setzen wollen, das momentan nicht geladen ist, und TDW diesen Ausdruck nicht zuordnen kann, können Sie entweder die Datei im Gültigkeitsbereich vorgeben, in der der Ausdruck enthalten ist, oder mit Hilfe von *View/Module* das aktuelle Modul wechseln.

Wenn Sie lokale Variablen als Bestandteil eines Ausdrucks verwenden, wird

der Breakpoint deutlich langsamer ausgeführt als ein Breakpoint, der sich lediglich auf globale oder modulspezifische Variablen bezieht.

Hardware-Breakpoints

Bei einem Hardware-Breakpoint werden die zusätzlichen Möglichkeiten von Hardware-Debuggern ausgenützt – entweder einer Hardware-Debugger-Platine oder der speziellen Debugger-Register des 80386-Prozessors (oder eines Nachfolgeprozessors). Wenn Ihr System für Hardware-Debuggen konfiguriert ist (*File/Get Info* zeigt Ihnen, ob die Breakpoints auf Hardware eingestellt sind), können Sie einen Hardware-Breakpoint mit einer der nachfolgenden Methoden setzen (In Kapitel 15, »Virtuell Debuggen mit dem 80386«, finden Sie eine Beschreibung, wie Sie die Gerätetreiber für Hardware-Debuggen konfigurieren können):

- Wählen Sie *Breakpoints/Changed Memory Global* , der häufigste Einsatzbereich für Hardware-Breakpoints.

- Wählen Sie *Breakpoints/Hardware* .

- Zeigen Sie das Dialogfenster *Breakpoints Options* an (mit Hilfe von *Breakpoints/At* oder dem Befehl *Set Options* im lokalen Menü des Fensters *View/Breakpoints*), und gehen Sie dann wie folgt vor:

1. Markieren Sie das Feld *Global* .

2. Wählen Sie den Schalter *Change* aus.

3. Im Dialogfenster *Conditions and Actions* schalten Sie im unteren Bereich des Dialogfensters auf *Hardware* um.

4. Drücken Sie den Schalter *Hardware* , um das Dialogfenster *Hardware Breakpoint Options* anzuzeigen.

5. Wählen Sie im Dialogfenster die gewünschten Optionen aus. Die Optionen sind in der Datei HDWDEBUG.TD beschrieben.

Variablenwerte protokollieren

Hin und wieder ist es praktisch, wenn man die Werte gewisser Variablen an bestimmten Programmstellen protokolliert. Sie können das Ergebnis von beliebigen Ausdrücken protokollieren, zum Beispiel auch die Parameter, mit denen eine Funktion aufgerufen wird. Wenn Sie sich das Protokoll mit allen Funktionsaufrufen anschauen, können Sie leicht herausfinden, wann die Funktion mit unzulässigen Parametern aufgerufen wurde. (Weitere Informationen dazu finden Sie im nachfolgenden Abschnitten über das Log-Fenster.)

Schalten Sie im Dialogfenster *Breakpoint Options* auf *Log* um. Daraufhin geben

Sie im Eingabefeld den Ausdruck ein, dessen Wert Sie jedesmal dann protokollieren wollen, wenn der Breakpoint ausgelöst wird. (Achten Sie bitte auf mögliche Seiteneffekte beim Protokollieren von Ausdrücken.)

Breakpoints und Templates

TDW unterstützt Breakpoints für C++ Templates. Die Breakpoints werden unterschiedlich gesetzt, je nachdem, ob Sie F2 im *Module*-Fenster, F2 im CPU-Fenster oder das Dialogfenster *Breakpoint Options* zum Setzen des Breakpoints verwenden.

Breakpoints bei Klassen-Templates

Es gibt mehrere Methodem um Breakpoints für Templates zu setzen:

- Wenn Sie einen Breakpoint in das Template selbst setzen, indem Sie F2 drücken, während sich der Cursor in der Programmzeile eines Template-Codes des *Module*-Fensters befindet, wird der Breakpoint für alle Klasseninstanzen dieses Templates gesetzt. Damit können Sie das Verhalten des Templates umfassend prüfen.

- Falls Sie einen Breakpoint in das Template mit Hilfe von Alt F2 im Dialogfenster *Breakpoint Options* setzen, erscheint bei der Eingabe der Adresse des Template-Ausdrucks ein Dialogfenster, in dem Sie die Klasseninstanz auswählen können, für die ein Breakpoint gesetzt werden soll.

- Wenn Sie das *CPU*-Fenster öffnen, können Sie erkennen, wo der Code der Klasseninstanz eines Templates vorkommt. Falls Sie nun den Cursor in eine Zeile des Template-Codes einer bestimmten Klasseninstanz setzen und F2 drücken, wird der Breakpoint nur für diese Klasseninstanz gesetzt.

Einen Breakpoint in einem Tempalte können Sie wie jeden anderen Breakpoint auch entfernen, indem Sie den Cursor im *Module*-Fenster auf die hervorgehobene Programmzeile setzen, F2 drücken oder den Löschbefehl des *Breakpoints*-Fensters verwenden. Dadurch werden auch alle damit eventuell verbundenen Breakpoints anderer Klasseninstanzen gelöscht.

Wenn Sie den Cursor im *CPU*-Fenster auf einen Breakpoint in einer Klasseninstanz setzen und F2 drücken, wird nur der Breakpoint der ausgewählten Klasseninstanz entfernt.

Breakpoints in Funktions-Templates

Breakpoints in Funktions-Tempaltes lassen sich in gleicher Weise setzen oder entfernen wie in Klassen-Templates. Die beiden Methoden zum Setzen von Breakpoints, Drücken von F2 oder Eingabe im Dialogfenster *Breakpoint Options* , funktionieren genauso wie bei Klasseninstanzen.

Breakpoints für Objekte und Template-Klasseninstanzen
Breakpoints für Objekte und Template-Klasseninstanzen setzen Sie wie bei normalen Klassen oder Objekte.

Das Fenster Log

Das *Log*-Fenster wird durch die Auswahl des Befehls *View/Log* geöffnet. Dieses Fenster ermöglicht ein Protokoll der wichtigsten Ereignisse, die während eines Debuggerlaufs eingetreten sind.

Abbildung 7.6
Das Log-Fenster

Das *Log*-Fenster zeigt eine Liste von Einträgen auf dem Bildschirm. Wenn mehr als 50 Zeilen in das *Log*-Fenster geschrieben wurden, werden die oberen Zeilen aus der Liste entfernt. Um die Anzahl der Zeilen zu verändern, können Sie entweder eine Kommandozeilenoption beim Aufruf verwenden oder die Anzahl mit Hilfe des Konfigurationsprogramms TWDINST dauerhaft einstellen. Sie können das gesamte *Log*-Fenster sichern, indem Sie den Inhalt des Fensters mit Hilfe des Befehls *Open log file* fortlaufend in eine Datei schreiben.

Mit folgenden Maßnahmen kann man Informationen in ein *Log*-Fenster schreiben:

- Ihr Programm hält an der von Ihnen angegebenen Stelle an. Diese Position wird in der *Log*-Datei protokolliert.

- Sie wählen den Befehl *Add comment* im lokalen Menü aus. Anschließend werden Sie nach dem Kommentar gefragt, den Sie im *Log*-Fenster aufnehmen wollen.

- Ein Breakpoint wird ausgelöst, der den Wert eines Ausdrucks in das *Log*-Fenster schreibt. Dieser Wert erscheint im Protokoll.

- Sie wählen den Befehl *Window/Dump pane to log* in der Menüleiste aus, um den aktuellen Inhalt eines Fensterausschnitts aufzuzeichnen.

- Sie debuggen eine MS-Windows-Anwendung und verwenden den Befehl *Display Windows Info* aus dem lokalen Menü des *Log*-Fensters, um Informationen über den globalen Heap, Informationen über den lokalen Heap oder die Modulliste zu protokollieren.

■ Sie debuggen eine MS-Windows-Anwendung und haben mit dem Befehl *View/Windows Messages* das Fenster *Windows Messages* aufgerufen, in dessen lokalen Menü Sie sich jetzt befinden. Sie schalten *Send to Log Window* auf *Yes* um, so daß alle Meldungen, die für dieses Fenster eingehen, auch zum *Log*-Fenster übertragen werden.

Das lokale Menü des Log-Fensters

Mit den Befehlen in diesem Menü kann man das Protokoll in eine Datei schreiben, das Protokoll abbrechen und wieder aufnehmen, Kommentare einfügen und das *Log*-Fenster löschen.

Mit der Tastenkombination Alt F10 wird das lokale Menü des *Log*-Fensters aufgerufen. Wenn die Strg -Tastenkürzel aktiviert sind, können Sie die Befehle direkt eingeben.

Open log file...

Der Befehl *Open log file...* (engl. Log-Datei öffnen) bewirkt, daß der Text, der im *Log*-Fenster erscheint, auch in eine Datei geschrieben wird. Sie werden anschließend nach dem Namen der Datei gefragt, in die das Protokoll geschrieben werden soll. Wenn Sie eine Log-Datei öffnen, dann werden alle Zeilen, die bereits im *Log*-Fenster dargestellt sind, in die Datei geschrieben. Daher kann eine Log-Datei auch dann noch geöffnet werden, nachdem man etwas Interessantes im *Log*-Fenster entdeckt hat, das in einer Datei gespeichert werden soll.

Wenn Sie beim Start des Dateiprotokolls die Zeilen, die bereits im *Log*-Fenster stehen, nicht in der Datei speichern wollen, dann müssen Sie vor der Auswahl von *Open log file* den Befehl *Erase log* angeben.

Close log file

Der Befehl *Close log file* (engl. Log-Datei schließen) beendet den Mitschnitt des *Log*-Fensters in der Datei, die mit dem lokalen Menübefehl *Open log file* ausgewählt wurde. Die Datei wird anschließend geschlossen.

Logging

Der Befehl *Logging* (engl. protokollieren) schaltet das Protokoll ein oder aus. Damit steuern Sie, was tatsächlich im *Log*-Fenster erscheinen soll.

Add comment

Der Befehl *Add comment* (engl. Kommentar einfügen) ermöglicht die Aufnahme von Kommentaren ins Protokoll. Sie können eine Textzeile Ihrer Wahl eingeben.

Erase log

Der Befehl *Erase log* (engl. Protokoll löschen) löscht das Protokoll. Das *Log*-Fenster ist daraufhin leer. Dieser Befehl beeinflußt aber nicht das Protokoll in der Datei.

Display Windows Info

Dieser Befehl öffnet das Dialogfenster *Windows Information* , mit dem Sie Informationen über den globalen Heap, den lokalen Heap und eine Liste der Module, aus denen Ihre Anwendung besteht, erfassen können. In Kapitel 11, »Debuggen unter Windows«, wird dieser Befehl genauer beschrieben.

8. Dateien untersuchen und ändern

Für Turbo Debugger für Windows sind Dateien nichts anderes als selbstverständliche Erweiterung des Programms, das Sie debuggen. Sie können jede beliebige Datei auf Ihrem Laufwerk untersuchen und deren Datein sowohl als ASCII-Text als auch in hexadezimaler Form dargestellen.

In diesem Kapitel wird gezeigt, wie man zwei Arten von Dateien untersuchen kann: Quelldateien eines Programms und beliebige andere Dateien.

Quelldateien eines Programms untersuchen

Die Quelldateien eines Programms sind die Dateien, die compiliert und dann zu einer »EXE«-Datei gelinkt werden. Sie werden diese Dateien in der Regel untersuchen wollen, wenn Sie das Verhalten oder die Struktur eines bestimmten Code-Abschnitts betrachten wollen. Beim Debuggen muß man oft im Quellcode einer Funktion nachschauen, weil man überprüfen will, ob die Argumente zulässig sind, oder ob die Funktion das korrekte Ergebnis zurückliefert.

Wenn Sie Ihr Programm schrittweise ausführen, dann zeigt TDW automatisch immer die aktuelle Position in der Quelldatei Ihres Programms an. Dateien, die mittels einer entsprechenden Compiler-Anweisung, wie zum Beispiel `#include` in C oder `INCLUDE` in Assembler, in Ihren Programmtext aufgenommen wurden, werden ebenfalls als Quelldateien behandelt, erscheinen jedoch nicht in der Listenauswahl von *View/Module*. Um eine dieser Dateien anzusehen, verwenden Sie den Befehl *File/Open*. (Informationen darüber wie eine Windows-DLL geladen und untersucht wird, erhalten Sie in Kapitel 11, »Debuggen unter Windows«.)

Sie sollten immer ein *Module*-Fenster geöffnet haben, damit Sie sich Ihre Quelldateien anschauen können, denn dadurch weiß TDW, daß ein Modul eine Quelldatei ist. Dadurch ist es sehr viel leichter, Breakpoints zu setzen oder Variablen zu untersuchen. Sie brauchen nur den Cursor auf die entsprechende Stelle in der Datei zu positionieren. Diese und andere Methoden werden wir im folgenden Abschnitt beschreiben.

Das Fenster Module

Vor dem Öffnen eines *Module*-Fensters müssen Sie ein Programm laden. Sie können ein *Module*-Fenster mit dem Befehl *View/Module* von der Menüleiste aus öffnen (oder Sie geben einfach das Tastenkürzel F3 ein).

Sie erhalten daraufhin ein Dialogfenster mit einer Liste von Modulen, aus der Sie das Modul heraussuchen können, das Sie betrachten wollen.

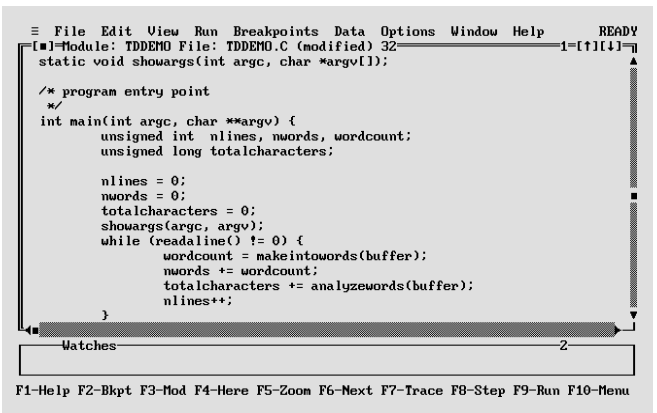


Abbildung 8.1 Das Module-Fenster

Wenn Sie eine Windows-Anwendung debuggen, zeigt dieses Dialogfenster eine Liste der Programm-Module und eine zweite Liste für DLLs und .EXE-Dateien, die momentan von Windows geladen sind.

TDW lädt das Modul, das Sie ausgewählt haben. Die Quelldateien werden in folgenden Verzeichnissen gesucht:

1. im Verzeichnis, in dem der Compiler die Programmdateien bei der Compilierung gefunden hat;

2. in den Verzeichnissen, die mit dem Befehl *Options/Path for source* oder der Kommandozeilenoption *-sd* angegeben wurden;

3. im aktuellen Verzeichnis;

4. im Verzeichnis, das das Programm enthält, das Sie debuggen.

Das *Module*-Fenster stellt den Inhalt der Quelldatei des von Ihnen ausgewählten Moduls dar. Die Überschrift des *Module*-Fensters enthält den Namen des gerade betrachteten Moduls zusammen mit dem Namen der Quelldatei und der Zeilennummer, in der der Cursor steht. Ein Pfeil in der ersten Spalte des Fensters zeigt die aktuelle Position im Programm (CS:IP) an.

Wenn die Abkürzung *opt* hinter dem Namen der Datei im Fenstertitel erscheint, wurde das Programm bei der Compilierung optimiert. Es kann sein, daß Sie einige Variablen nicht mehr finden, da sie der Optimierung zum Opfer fielen. Außerdem sollten Sie beachten, daß Variablen, die in Registervariablen umgewandelt wurden, keine Adresse besitzen.

Wenn die Bezeichnung *modified* (verändert) im Fenstertitel hinter dem Dateinamen auftaucht, dann bedeutet das, daß die Quelldatei seit dem letzten Compilieren oder Linken des Programms verändert wurde. Daraus folgt unter Umständen, daß die Routinen in den geänderten Quelldateien nicht mehr dieselben Zeilennummern aufweisen wie die Version, die zum Erstellen des Programms, das Sie gerade debuggen, verwendet wurde. Dadurch kann der Pfeil, der zur Anzeige der aktuellen Position (CS:IP) dient, auf eine falsche Zeile weisen.

Das lokale Menü des Module-Fensters

Das lokale Menü des *Module*-Fensters enthält eine Reihe von Befehlen, die dazu dienen, den Cursor im dargestellten Modul zu positionieren, Datenobjekte zu untersuchen und das Fenster zur Darstellung anderer Dateien oder Module zu verwenden.

Sie werden dieses Menü wahrscheinlich öfter als jedes andere Menü in TDW verwenden, so daß Sie mit den verschiedenen Optionen gut vertraut sein sollten.

Mit der Tastenkombination Alt F10 öffnen Sie das lokale Menü des *Module*-Fensters. Sind die Strg -Tastenkürzel aktiviert, können Sie die einzelnen Befehle auch mit Strg und dem Anfangsbuchstaben des gewünschten Befehls aufrufen.

Inspect

Der Befehl *Inspect* (engl. untersuchen) öffnet ein Fenster, in dem Sie den Inhalt der Programmvariablen an der aktuellen Cursor-Position untersuchen können. Bevor Sie diesen Befehl aufrufen, müssen Sie den Cursor auf eine Programmvariable in der Quelldatei setzen.

Sie können auch die Taste Einfg dazu verwenden, um einen Ausdruck auszuwählen, der dann hervorgehoben wird. Damit ersparen Sie sich das Eintippen eines Ausdrucks, der ohnehin im Quelltext steht.

Da Sie mit diesem Befehl nicht mehr jeden Namen einzeln eingeben müssen, werden Sie diesen Befehl wahrscheinlich sehr häufig einsetzen, um einzelne Variablenwerte zu kontrollieren.

Watch

Der Befehl *Watch* (engl. beobachten) fügt die Variable an der aktuellen Cursor-Position ins *Watches* -Fenster ein. Das ist sehr praktisch, wenn man den Inhalt einer Variablen während der Programmausführung die ganze Zeit über im Auge haben will. Bevor Sie diesen Befehl aufrufen, müssen Sie den Cursor auf die zur Programmvariable in der Quelldatei setzen.

Mit der Taste Einfg können Sie einen Ausdruck direkt in das *Watches* -Fenster aufnehmen, ohne ihn dazu neu eintippen zu müssen.

Module...

Mit diesem Befehl kann ein anderes Modul in das aktuelle *Module*-Fenster geladen werden. Verwenden Sie diesen Befehl, wenn Sie am aktuellen Modul

nicht mehr weiter interessiert sind und nicht mehrere *Module*-Fenster auf dem Bildschirm öffnen wollen.

File

Der Befehl *File* ermöglicht die Darstellung einer anderen, dem aktuellen Modul zugeordneten Quelldatei. Wählen Sie die Datei, die dargestellt werden soll, aus der angebotenen Dateiliste aus. Die meisten Module bestehen nur aus einer einzigen Quelldatei. In den anderen Dateien eines Moduls werden in der Regel nur Konstanten und Datenstrukturen definiert. Sie sollten diesen Befehl verwenden, wenn sich Ihr Modul aus mehr als einer Quelldatei zusammensetzt.

Um die erste Datei anzusehen, verwenden Sie den Befehl *View/File*. Wenn Sie mehr als eine Datei gleichzeitig auf dem Bildschirm darstellen wollen, müssen Sie den Befehl *View/Another/File* über die Menüleiste auswählen.

Previous

Mit dem Befehl *Previous* (engl. vorherige Position) kommen Sie zu der Position in Ihrer Quelldatei zurück, die Sie zuletzt betrachtet haben. Sie können diesen Befehl auch dazu verwenden, um zur vorherigen Position zurückzukehren, wenn Sie einen Befehl eingegeben haben, der die Position im bearbeiteten Modul verändert hat.

Line...

Mit dem Befehl *Line* (engl. Zeile) können Sie den Cursor auf eine bestimmte Zeile in Ihrer Datei positionieren. Wenn Sie eine Zeilennummer eingeben, die hinter der letzten Zeile in der Datei liegt, dann wird die letzte Zeile der Datei im Fenster dargestellt.

Search

Der Befehl *Search* (engl. suchen) sucht ab der aktuellen Cursor-Position einen String. Geben Sie dazu die Zeichenfolge ein, nach der gesucht werden soll. Wenn der Cursor auf einem Bezeichner steht, dann wird das Eingabefeld des Such-String mit diesem Bezeichner initialisiert. Haben Sie einen Block in der Datei mit der Taste Einfg markiert, dann wird Ihnen dieser Block zur Übernahme angeboten, was Ihnen vielleicht unnötige Tipparbeit erspart.

Sie können auch einfache Jokerzeichen verwenden. Das Fragezeichen ? steht für ein einzelnes Zeichen und der Stern * für null und mehr Zeichen im gesuchten String. Die Suche wird nur bis zum Ende der Datei durchgeführt und fängt nicht wieder am Dateianfang an. Um die gesamte Datei zu durchsuchen, müssen Sie mit Strg Bild↑ den Cursor in die erste Zeile der Datei setzen.

Next

Der Befehl *Next* (engl. Nächster) sucht das nächste Vorkommen des Such-String, der mit dem letzten *Search*-Befehl eingegeben wurde. Der Befehl kann nur dann aufgerufen werden, wenn zuvor bereits ein Suchbefehl durchgeführt wurde.

Manchmal findet man mit einem Suchbefehl nicht das gewünschte Vorkommen des Such-String. Mit *Next* kann man die Suche wiederholen, ohne den String noch einmal eingeben zu müssen.

Origin

Mit dem Befehl *Origin* (engl. Ursprung) gelangen Sie zu der Zeile eines Moduls, die der augenblicklich mit CS:IP adressierten Programmposition entspricht. Wenn das Modul, das Sie gerade dargestellt haben, nicht das Modul mit der aktuellen Programmposition ist, wird das entsprechende Modul in das *Module*-Fenster geladen. Dieser Befehl ist sehr nützlich, wenn Sie zwischendurch verschiedene Teile Ihres Programms ansehen wollen, anschließend jedoch wieder zur aktuellen Position zurückzukehren wollen.

Goto...

Der Befehl *Goto* (engl. gehe zu) bringt Sie an jede beliebige Stelle in Ihrem Programm. Geben Sie die Adresse an, die Sie untersuchen wollen. Sie können dazu einen Funktionsnamen oder eine Hexadezimaladresse angeben. In Kapitel 9, »Ausdrücke«, finden Sie eine vollständige Beschreibung der Möglichkeiten, eine Adresse einzugeben.

Sie können diesen Befehl auch ganz einfach aufrufen, indem Sie nur den Anfang des Labels, das Sie anschauen wollen, eingeben. TDW öffnet daraufhin ein Dialogfenster. Das ist eine recht praktische Kurzform für einen sehr häufig verwendeten Befehl. (Falls für die Adresse keine entsprechende Quellzeile vorliegt, wird ein *CPU*-Fenster geöffnet.)

Andere Dateien untersuchen

Mit Hilfe des *File*-Fensters können Sie beliebige Dateien untersuchen. Die Dateien werden entweder als ASCII-Text oder hexadezimaler Form dargestellt. Dies kann mit den Schaltern *Ascii* und *Hex* des Befehls *Display as* eingestellt werden, die wir in einem der nachfolgenden Abschnitte besprechen.

Das Fenster File

Mit Hilfe des Befehls *View/File* können Sie ein *File*-Fenster öffnen. Geben Sie den genauen Namen der Datei ein, die geladen werden soll, oder verwenden Sie die DOS-Jokerzeichen, um eine Dateiliste zu erhalten.

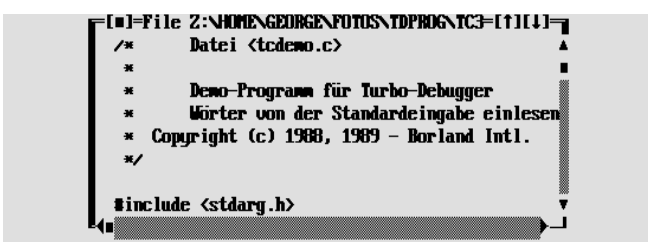


Abbildung 8.2
Das File-Fenster

Die *File*-Fenster stellen den Inhalt der ausgewählten Datei dar. Der Name der dargestellten Datei erscheint im Falle einer ASCII-Datei zusammen mit der aktuellen Zeilennummer in der Fensterüberschrift.

Wenn Sie ein *File*-Fenster zum ersten Mal öffnen, dann erscheint der Inhalt der Datei entweder als ASCII-Text oder in hexadezimaler Darstellung. Das hängt davon ab, ob TDW den Inhalt der Datei als ASCII-Text oder als Binärdaten interpretiert. Sie können jedoch zwischen der ASCII- und der Hexadezimaldarstellung jederzeit mit dem Befehl *Display as* umschalten.

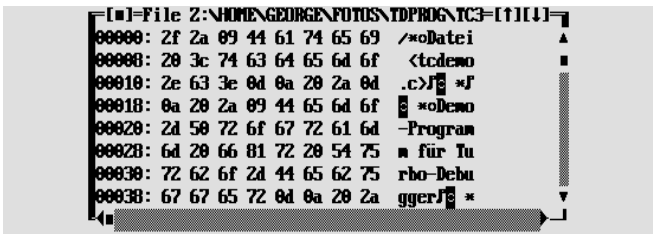


Abbildung 8.3
Das Datei-Fenster mit Hexadezimal-Darstellung

Das lokale Menü des File-Fensters

Das lokale Menü des *File*-Fensters enthält eine Reihe von Befehlen, um bestimmte Ausschnitte einer Datei darzustellen, Veränderung der Darstellung vorzunehmen und die Datei selbst zu modifizieren.

Mit der Tastenkombination Alt F10 öffnen Sie das lokale Menü des *File*-Fensters. Sind die Strg -Tastenkürzel aktiviert, können Sie die einzelnen Befehle auch mit Strg und dem Anfangsbuchstaben des gewünschten Befehls aufrufen.

Goto

Der Befehl *Goto* (engl. gehe zu) versetzt den Cursor direkt in eine neue Zeile oder an einen neuen Offset in der Datei. Wenn Sie die ASCII-Darstellung gewählt haben, dann geben Sie die Nummer der Zeile ein, die Sie sehen wollen. Wenn die Datei hexadezimal dargestellt wird, dann müssen Sie den Offset der gewünschten Position ab dem Dateianfang angeben. Sie können den Offset in allen Schreibweisen, die vom Parser unterstützt werden, eingeben. Wenn Sie eine Zeilennummer oder einen Offset eingeben, der hinter dem Dateiende liegt, wird das Dateiende dargestellt.

Search

Der Befehl *Search* (engl. suchen) sucht ab der aktuellen Position nach einem String. Dieser Befehl öffnet ein Dialogfenster, in das Sie den gesuchten String eingeben müssen. Wenn der Cursor auf einem Symbolbezeichner steht, dann wird das Eingabefeld des Such-String mit diesem Bezeichner initialisiert. Haben Sie einen Block in der Datei mit der Taste Einfg markiert, erhalten Sie den Inhalt des Blocks als erste Vorgabe für den Such-String. Auf diese Weise können Sie sich viel Tipparbeit sparen. Das Format des Such-String hängt davon ab, ob die Datei im ASCII- oder im Hex-Modus dargestellt wird.

Wenn die Datei im ASCII-Modus dargestellt wird, dann können Sie einfache Jokerzeichen verwenden. Das Fragezeichen ? steht für ein einzelnes Zeichen, der Stern (*) für null und mehr Zeichen.

Wird die Datei im Hex-Modus dargestellt, dann können Sie entweder eine Byte-Folge oder einen String in Anführungszeichen eingeben. Sie können den String in der Syntax angeben, die von einer der verfügbaren Sprachen unterstützt wird. In Kapitel 9, »Ausdrücke«, finden Sie eine ausführliche Darstellung der Byte-Folgen.

In C++ wird eine Byte-Folge für die Hexadezimalzahl 0408 wie folgt eingegeben:

```
0x0804
```

In Pascal wird diese Byte-Folge wie folgt eingegeben:

```
$0804
```

Die Suche wird nur bis zum Ende der Datei durchgeführt und fängt nicht wieder am Dateianfang an. Um die gesamte Datei zu durchsuchen, müssen Sie den Cursor mit Strg Bild↑ in die erste Zeile der Datei setzen.

Sie können den Sachbefehl auch dadurch aktivieren, daß Sie den String, den Sie suchen, einfach direkt eintippen. Daraufhin wird ein Dialogfenster geöffnet, in dem Sie den Such-String weiter eintippen können.

Next

Der Befehl *Next* (engl. Nächster) sucht das nächste Vorkommen des Such-String, der mit dem letzten *Search*-Befehl eingegeben wurde. Der Befehl kann nur dann aufgerufen werden, wenn zuvor bereits ein Suchbefehl durchgeführt wurde.

Manchmal findet man mit einem Suchbefehl nicht das gewünschte Vorkommen des Such-String. Mit *Next* kann man die Suche wiederholen, ohne den String noch einmal eingeben zu müssen.

Display as

Der Befehl *Display as* (engl. dargestellen als) schaltet die Darstellungsart der Datei um. Die Datei kann entweder als ASCII-Text oder hexadezimal dargestellt werden. Wenn Sie die ASCII-Darstellung wählen, dann erscheint die Datei so, wie Sie es von Ihrem Editor oder Ihrer Textverarbeitung gewöhnt sind. Wenn Sie die Hex-Darstellung wählen, dann steht am Anfang jeder Zeile in hexadezimaler Darstellung die Entfernung der aktuellen Position vom Dateianfang. Es werden entweder 8 oder 16 Byte in einer Zeile dargestellt. Dies hängt von der Breite des Fensterausschnitts ab. Auf der rechten Seite der *Hex*-Darstellung werden die ASCII-Entsprechungen der dargestellten Byte aufgeführt. Dabei wird der vollständige Zeichensatz verwendet, so daß auch die Werte unter 32 und über 127 mit dem entsprechenden Symbol dargestellt werden.

File...

Mit dem Befehl *File...* kann man eine andere Datei in das *File*-Fenster laden. Sie können entweder die DOS-Jokerzeichen verwenden, um eine Dateiliste zur Auswahl zu erhalten, oder den Namen der Datei, die geladen werden soll, direkt

eingeben. Wenn Sie zwei verschiedene Dateien oder zwei unterschiedliche Abschnitte einer Datei gleichzeitig auf dem Bildschirm darstellen wollen, dann können Sie den Befehl *View/Another/File* angeben, um ein weiteres *File*-Fenster zu öffnen.

9. Ausdrücke

Ausdrücke können eine Kombination von Symbolen Ihres Programms (also Variablen- und Routinenbezeichner) sowie Konstanten und Operatoren aus einer der unterstützten Programmiersprachen C, Pascal oder Assembler sein.

TDW kann Ausdrücke auswerten und Ihnen das Ergebnis mitteilen. Sie können Ausdrücke auch für die Bezeichnung eines Datenbereichs im Speicher verwenden, dessen Wert Sie ermitteln wollen. Sie können einen Ausdruck überall dort angeben, wo ein Wert oder eine Speicheradresse angefordert wird. (Beachten Sie, daß jede Programmiersprache einen Ausdruck unterschiedlich auswertet.)

Mit dem Befehl *Data/Evaluate/Modify* können Sie den Wert eines Ausdrucks über die Menüleiste ermitteln. Sie können diesen Befehl auch für die Abfrage eines einfachen Datenelements verwenden, sowie zum Untersuchen der Werte in Ihren Programmdaten einsetzen. Sie erhalten daraufhin ein Dialogfenster, in dem Sie nach einem Wert oder einer Speicheradresse gefragt werden.

In diesem Kapitel werden Sie lernen, wie TDW die Programmiersprache auswählt, mit der ein Ausdruck ausgewertet wird, und wie man eine bestimmte Programmiersprache vorgeben kann. Wir beschreiben die Bestandteile der Ausdrücke, die allen Programmiersprachen gemeinsam sind, zum Beispiel die Zeilennummern in der Quelldatei und der Zugriff auf die Register des Prozessors. Anschließend werden die für jede Programmiersprache spezifischen Komponenten der Ausdrücke behandelt. Dazu zählen Konstanten, Programmvariablen, Strings und Operatoren. Für jede Sprache geben wir die Operatoren und die Syntax an, die TDW unterstützt.

Eine komplette Behandlung der Ausdrücke in C, C++ und Assembler finden Sie in Ihren Benutzer- und Referenzhandbüchern von Turbo C++ für Windows.

Die Auswahl der Sprache für die Auswertung der Ausdrücke

TDW bestimmt, in welcher Sprache die Ausdrücke ausgewertet werden sollen, normalerweise anhand der Dateierweiterung des aktuellen Quellmoduls. Dies ist das Modul, in dem Ihr Programm angehalten wurde. Sie können diese Einstellung aber mit dem Befehl *Options/Language* ändern, indem Sie immer Source, C, Pascal oder Assembler im Dialogfenster vorgeben. Wenn Sie die

Option *Source* auswählen, werden die Ausdrücke entsprechend dem Quellmodul ausgewertet. (Falls TDW die Programmiersprache nicht kennt, werden die C-Regeln in den Ausdrücken verwendet.)

Üblicherweise überläßt man TDW die Auswahl der Sprache. Manchmal ist es jedoch praktisch, die Programmiersprache ausdrücklich vorzugeben, zum Beispiel beim Debuggen eines Assembler-Moduls, das von einer anderen Sprache aufgerufen wird. Durch die ausdrückliche Festlegung auf eine bestimmte Programmiersprache bei der Auswertung der Ausdrücke kann man auf die Programmdaten mit dieser Sprache zugreifen, auch wenn das aktuelle Modul eine andere Sprache verwendet.

Manchmal ist es sehr praktisch, die Ausdrücke oder Variablen so zu behandeln, als wären Sie in einer anderen Sprache verfaßt. Wenn Sie zum Beispiel ein C++Programm debuggen, kann es sein, daß man den Wert eines Byte in einem String mit den Assembler-Konventionen leichter ändern kann.

Solange die erste Spracheinstellung beim Aufruf von TDW in Ordnung ist, sollten Sie keine Probleme haben, andere Sprachkonventionen anzuwenden. TDW behält die Informationen der ursprünglichen Quellsprache und verarbeitet die Umwandlungen und Speicherbefehle korrekt. Wenn ein Ausdruck mehrere Deutungen zuläßt, greift TDW auf die Assembler-Sprache zurück.

Auch wenn Sie beim Aufruf von TDW absichtlich die falsche Sprache ausgewählt haben, gibt es immer noch die Möglichkeit, einige Informationen über die ursprüngliche Quellsprache aus der Symboltabelle der Quelldatei zu entnehmen. Unter gewissen Umständen kann es jedoch sein, daß TDW die Daten nicht richtig speichert.

Adressen im Code, Adressen von Daten, Zeilennummern

Wenn Sie auf eine Variable oder eine Funktion in Ihrem Programm zugreifen, geben Sie üblicherweise nur den entsprechenden Namen ein. Man kann jedoch auch einen Ausdruck eingeben, der einen Zeiger ergibt, oder eine Adresse im Programm als Zeilennummer angeben. Vor der Zeilennummer muß ein Nummernzeichen (#) stehen, wie zum Beispiel in #123. Im nächsten Abschnitt beschreiben wir, wie man auf Symbole zugreifen kann, die außerhalb des aktuellen Gültigkeitsbereichs liegen.

Eine reguläre Segemnt:Offset-Adresse können Sie auch in hexadezimaler Schreibweise der Quellcode-Syntax eingeben:

Sprache	Format	Beispiel
C	0x*nnnn*	0x1234:0x0010
Assembler	*nnnn*h	1234h:0010h 1234h:0B234h

In Assembler muß vor Hexadezimalzahlen, die mit den Buchstaben A bis F beginnen, eine Null stehen.

Der Zugriff auf Symbole außerhalb des Gültigkeitsbereichs

Den Bereich, in dem der Debugger nach einem Symbol sucht, bezeichnen wir als aktuellen *Gültigkeitsbereich* dieses Symbols. Der Zugriff auf Symbole außerhalb des aktuellen Gültigkeitsbereichs ist ein erweitertes Konzept, dessen spezielle Kenntnis bei der Arbeit mit TDW meist nicht unbedingt erforderlich ist.

Normalerweise sucht TDW ein Symbol in einem Ausdruck auf dieselbe Art und Weise wie der Compiler.

In C wird ein Symbol zum Beispiel zuerst in der aktuellen Funktion als lokales Symbol gesucht, dann im aktuellen Modul als statisches Symbol und schließlich als globales Symbol. In Pascal wird das Symbol zunächst in der aktuellen Prozedur oder Funktion gesucht, dann in einem der äußeren Unterprogramme (wenn das Unterprogramm in ein anderes Unterprogramm verschachtelt ist), anschließend im Implementation-Abschnitt der aktuellen Unit (falls der aktuelle Gültigkeitsbereich in einer Unit liegt) und schließlich als globales Symbol.

Wenn TDW das Symbol mit dieser Vorgehensweise nicht finden kann, werden alle anderen Module nach einem entsprechenden statischen Symbol durchsucht. Damit kann man sich auf Symbole in anderen Modulen beziehen, ohne den Modulnamen angeben zu müssen.

Wird TDW gezwungen, das Symbol anderswo zu suchen, dann kann man die gesamte Suche exakt steuern, indem man ein Modul, eine Datei in einem Modul oder eine Routine vorgibt, in der das Symbol gesucht werden soll. Damit ist der Zugriff auf jedes Symbol, das einen definierten Wert hat, in Ihrem Programm gewährleistet, auch wenn die Symbole nur von diesen Prozeduren oder Funktionen verwendet werden dürfen und die Bezeichnungen mit anderen Symbolen verwechselt werden können.

Die Vorgabe eines Gültigkeitsbereichs

Je nach verwendeter Programmiersprache verwenden Sie unterschiedliche Symbole, um den Gültigkeitsbereich für ein Symbol vorzugeben:

- In C, C++ für Windows und Turbo Assembler verwenden Sie das Nummerzeichen #, um den Gültigkeitsbereich vorzugeben.
- In Pascal kann man den Gültigkeitsbereich durch einen Punkt (.) vorgeben.

Eine Vorgabe des Gültigkeitsbereichs kann bei allen Ausdrücken verwendet werden, also auch:

- im Dialogfenster *Evaluate/Modify* ;
- im *Watches* -Fenster;
- in einem *Data/Inspector* -Fenster;

- im Dialogfenster, das nach einem *Goto*-Befehl im *Module*-Fenster erscheint (wenn Sie an eine Adresse im Quellcode springen wollen).

Die Vorgabe des Gültigkeitsbereichs in C, C++ für Windows und Assembler

Im Regelfall verwendet man das Nummernzeichen (#), um die einzelnen Elemente eines Gültigkeitsbereichs voneinander zu trennen. (Bei Register-Variablen funktioniert die Vorgabe des Gültigkeitsbereichs nicht.)

Die folgende Syntax beschreibt, wie der Gültigkeitsbereich vorgegeben werden kann (optionale Parameter stehen in eckigen Klammern, [ ]):

```
[#Modul[#Dateiname]]#Zeilennummer[#Variable]
```

oder

```
[#Modul[#Dateiname]][#Funktion]#Variable
```

Wenn Sie kein spezielles Modul angeben, wird das aktuelle Modul verwendet. Im *Watches* -Fenster könnten Sie zum Beispiel verschiedene Zeilennummern angeben, um zu sehen, wie sich der Wert der Variablen `nlines` von TDDEMO in den verscheinden Routinen des aktuellen Moduls ändert. Um diese Variable zum Beispiel in den Zeilen 51 und 72 zu überwachen, schreiben Sie folgende Einträge in das das *Watches* -Fenster:

```
#51#nlines
#72#nlines
```

Es folgen einige Beispiele für zulässige Vorgaben des Gültigkeitsbereichs in Ausdrücken mit Symbolen. Es wurde jeweils ein Beispiel einer zulässigen Kombination für die Elemente dargestellt, um den Gültigkeitsbereich vorzugeben.

Die ersten sechs Beispiele zeigen die verschiedenen Möglichkeiten, wie man Zeilennummern für die Erzeugung von Adressen und Vorgaben des Gültigkeitsbereichs verwenden kann:

`#123`	Zeile 123 im aktuellen Modul.
`#123#Var1`	Das Symbol `Var1` in der Zeile 123 des aktuellen Moduls.
`#Mod1#123`	Zeile 123 im Modul `Mod1`.
`#Mod1#123#Var1`	Symbol `Var1` in der Zeile 123 des Moduls `Mod1`.
`#Mod1#Dat1#123`	Zeile 123 der Quelldatei `Dat1` des Moduls `Mod1`.
`#Mod1#Dat1#123#Var1`	Symbol `Var1` in Zeile 123 der Quelldatei `Dat1` des Moduls `Mod1`.

Die nächsten sechs Beispiele zeigen die verschiedenen Möglichkeiten, um den Gültigkeitsbereich einer Variaben mit Hilfe des Moduls, der Datei und der Funktionsbezeichnung vorzugeben:

`#Var2`	Variable `Var2` im aktuellen Modul.

`#Funk#Var2`	Variable `Var2` in der Funktion `Funk` des aktuellen Moduls.
`#Mod2#Var2`	Variable `Var2` im Modul `Mod2`.
`#Mod2#Funk#Var2`	Variable `Var2` in der Funktion `Funk`, die im Modul `Mod2` definiert ist.
`#Mod2#Dat2#Var2`	Variable `Var2` in der Datei `Dat2` des Moduls `Mod2`.
`#Mod2#Dat2#Funk#Var2`	Variable `Var2` in der Funktion `Funk`, die in der Datei `Dat2` des Moduls `Mod2` definiert ist.

Die folgenden vier Beispiele zeigen, wie Sie den Gültigkeitsbereich für Klassen, Objekte, Elementfunktionen und Datenelemente in C++ für Windows vorgeben können:

`Obj#ElemVar`	Datenelement `ElemVar` des Objekts `Obj` im aktuellen Gültigkeitsbereich.
`Obj#ElemFunk`	Elementfunktion `ElemFunk` des Objekts `Obj` innerhalb des aktuellen Gültigkeitsbereich.
`Mod#Obj#ElemVar`	Datenelement `ElemFunk` des Objekts `Obj` in Modul `Mod`.
`#Mod#Obj#ElemFunk`	Elementfunktion `ElemFunk` des Objekts `Obj` innerhalb von Modul `Mod`.

Wenn Sie ein Programm debuggen, das in Turbo C++ für Windows verfaßt wurde, und eine überladene Funktion untersuchen wollen, geben Sie nur den Namen der entsprechenden Funktion im Dialogfenster ein. TDW öffnet das Dialogfenster *Pick a Symbol Name* , in dem eine Liste mit allen Funktionen dieses Namens und den übergebenen Argumenten erscheint, aus der Sie die gewünschte Funktion auswählen können.

Hinweise für Gültigkeitsvorgaben

Die folgenden Hinweise helfen Ihnen wahrscheinlich bei der Gültigkeitsvorgabe in C, C++ für Windows oder Assembler-Programmen:

1. Wenn Sie einen Dateinamen in einer Gültigkeitsvorgabe verwenden, muß der Name des Moduls vorangestellt werden.

2. Wenn ein Dateiname eine bestimmte Dateierweiterung hat, wie .ASM, .C oder .CPP, müssen Sie diese Dateierweiterung angeben; TDW unternimmt keine Versuche, den vollständigen Dateinamen herauszufinden.

3. Wenn Sie einen Funktionsnamen als erstes Element in einer Vorgabe für den Gültigkeitsbereich verwenden, darf kein Nummernzeichen, »#«, davor stehen. Falls ein Nummernzeichen an erster Stelle steht, wird der Funktionsname als Modulname interpretiert.

4. Jede Variable, auf die Sie über eine Gültigkeitsvorgabe zugreifen, muß vorher intialisiert worden sein. Eine automatische Variable muß sich nicht innerhalb

des aktuellen Gültigkeitsbereichs befinden, aber die Funktion darf noch nicht aufgerufen worden sein.

5. Wenn Sie auf eine automatische Variable zugreifen wollen, die sich nicht mehr innerhalb des aktuellen Gültigkeitsbereichs befindet, müssen Sie den Funktionsnamen in die Vorgabe des Gültigkeitsbereichs aufnehmen.

6. TDW unterstützt Templates und verschachtelte Klassen. Der Gültigkeitsbereich von Templates hängt von der aktuellen Programmposition ab. *Watches* - und *Inspector*-Fenster für Template-Ausdrücke sind vom aktuellen Programmobjekt abhängig. Eine verschachtelte Klasse befindet sich im Gültigkeitsbereich der Klasse, der sie untergeordnet ist. Der Gültigkeitsbereich einer verschachtelten Klasse ist nicht global.

Der Gültigkeitsbereich und DLLs

Da TDW die Symboltabellen des aktuellen Moduls Ihrer .EXE-Datei und aller DLLs, auf die zugegriffen wird, automatisch lädt, haben Sie unter Umständen keinen direkten Zugriff auf die Variablen in Ihren DLLs (oder Ihrer .EXE-Datei), wenn Sie gerade eine DLL bearbeiten.

TDW sucht die Variablen in der Symboltabelle des aktuellen Moduls oder der aktuellen DLL, und erst dann in den anderen Symboltabellen in der Reihenfolge, in der sie geladen wurden. Wenn eine Variable in mehreren DLLs oder in Ihrer .EXE-Datei unter demselben Namen vorkommt, verwendet TDW nur die erste Instanz, die gefunden wird. Es nützt nichts, den Gültigkeitsbereich dieser Variablen vorzugeben. Sie müssen F3 drücken und das Dialogfenster *Load Modules and DLL* verwenden, um das entsprechende Modul oder die DLL zu laden.

TDW lädt die Symboltabellen für folgende Dateien:

1. das aktelle Modul Ihrer .EXE.Datei;

2. alle DLLs, die Sie mit dem Befehl *Symbol Load* im Dialogfenster *Load Modules and DLLs* laden (F3 oder *View/Module*);

3. Jede DLL, die Sie untersuchen.

Der implizite Gültigkeitsbereich für die Auswertung von Ausdrücken

Jedesmal, wenn ein Ausdruck von TDW ausgewertet wird, muß der aktuelle Gültigkeitsbereich für die Symbolbezeichnungen bestimmt werden, die keine expliziten Vorgaben für den Gültigkeitsbereich aufweisen. Das ist in vielen Programmiersprachen sehr wichtig, weil man Symbole, die denselben Namen tragen, in Funktionen oder Prozeduren wie ein globales Symbol verwenden kann. TDW muß genau wissen, welches Symbol Sie meinen.

Normalerweise verwendet TDW die aktuelle Cursorposition als Ausgangspunkt, um den Gültigkeitsbereich festzulegen. Sie können die Festlegung zum Beispiel vornehmen, indem Sie den Cursor auf eine bestimmten Zeile im *Module*-Fenster bewegen.

Das bedeutet, daß Sie unerwartete Ergebnisse erhalten können, wenn Sie den Cursor von der Zeile wegbewegen, an der Ihr Programm angehalten wurde. Wenn Sie sicherstellen wollen, daß die Ausdrücke im aktuellen Gültigkeitsbereich Ihres Programms ausgewertet werden, können Sie den lokalen Befehl *Origin* im *Module*-Fenster aufrufen, um wieder zur aktuellen Position in der Quelldatei zurückzukehren. Der Gültigkeitsbereich für die Ausdrücke wird auch dadurch verändert, indem man im Code-Ausschnitt des *CPU*-Fensters herumwandert, den Cursor im *Stack*-Fenster verschiebt oder im *Variables*-Fenster auf einen bestimmten Funktionsnamen stellt.

Byte-Folgen

In einigen Befehlen müssen Byte-Folgen eingegeben werden. Dazu gehören die lokalen Befehle *Search* und *Change* im Datenausschnitt des *CPU*-Fensters sowie die lokalen Befehle *Search* und *Change* im *File*-Fenster, wenn die Datei im Hexadezimalformat dargestellt wird.

Eine Byte-Folge kann eine beliebige Mischung aus skalaren Zahlen (keine Fließkommazahlen) und Strings sein, die in der Syntax der gerade ausgewählten Programmierspache angegeben werden müssen. Die Programmiersprache kann mit dem Befehl *Options/Language* eingestellt werden. Sowohl Strings als auch die skalaren Angaben verwenden dieselbe Syntax, die auch in den Ausdrücken vorkommt. Skalare Zahlen werden in die entsprechende Byte-Folge umgewandelt. Aus einem `LongInt` mit dem Wert `123456` in Pascal wird eine Folge von vier Bytes mit den Hexadezimalwerten 56 34 12 00 erzeugt.

Sprache	Byte-Folge	Hex-Daten
C	"ab" 0x04 "c"	61 62 04 63
Pascal	'ab'#4'c'	61 62 04 63
Assembler	1234 "AB"	34 12 41 42

C-Ausdrücke

TDW unterstützt alle in der C-Syntax erlaubten Ausdrücke. Ein Ausdruck besteht aus einer Kombination von Operatoren, Strings, Variablen und Konstanten. Jedes dieser Elemente wird in den folgenden Abschnitten näher beschrieben.

C-Symbole

Ein Symbol ist eine Bezeichnung für ein Datenobjekt oder eine Routine in Ihrem Programm. Eine Symbolbezeichnung wird durch einen Buchstaben oder einen Unterstrich eingeleitet. In den nachfolgenden Zeichen können sowohl diese Zeichen selbst, zusätzlich aber auch die Ziffern 0 bis 9 verwendet werden. Der einleitende Unterstrich kann bei den Symbolbezeichnungen weggelassen werden. Wenn Sie einen Symbolnamen ohne einen Unterstrich eingeben und der Name wird nicht gefunden, dann wird der Name noch einmal mit einem Unterstrich gesucht. Da der Compiler normalerweise einen Unterstrich am Anfang der Symbole verwendet, können Sie auf diese Angabe verzichten.

Pseudovariablen für die C-Register

TDW ermöglicht den Zugriff auf die Register des Prozessors mit derselben Methode wie die C++ Compiler von Borland: über die Pseudovariablen. Eine Pseudovariable ist eine Variablenbezeichnung für das entsprechende Register des Prozessors.

Pseudovariable	Datentyp	Register
_AX	unsigned int	AX
_AL	unsigned char	AL
_AH	unsigned char	AH
_BX	unsigned int	BX
_BL	unsigned char	BL
_BH	unsigned char	BH
_CX	unsigned int	CX
_CL	unsigned char	CL
_CH	unsigned char	CH
_DX	unsigned int	DX
_DL	unsigned char	DL
_DH	unsigned char	DH
_CS	unsigned int	CS
_DS	unsigned int	DS
_SS	unsigned int	SS
_ES	unsigned int	ES
_SP	unsigned int	SP
_BP	unsigned int	BP
_DI	unsigned int	DI
_SI	unsigned int	SI
_IP	unsigned int	Programmzeiger

Mit den folgenden Pseudovariablen kann man auf die Register des 80386-Prozessors zugreifen:

Pseudovariable	Datentyp	Register
_EAX	unsigned long	EAX
_EBX	unsigned long	EBX
_ECX	unsigned long	ECX
_EDX	unsigned long	EDX
_ESP	unsigned long	ESP
_EBP	unsigned long	EBP
_EDI	unsigned long	EDI
_ESI	unsigned long	ESI
_FS	unsigned int	FS
_GS	unsigned int	GS

C-Konstanten und Zahlenformate

Als Konstanten kann man entweder ganze Zahlen oder Fließkommazahlen verwenden. Eine Integer-Konstante wird dezimal angegeben, wenn man nicht eines der folgenden C-Formate vorgibt:

Format	Zahlenbasis
Ziffern	dezimal
0*Ziffern*	oktal
0X*Ziffern*	hexadezimal
0x*Ziffern*	hexadezimal

Die Konstanten werden normalerweise mit dem Datentyp `int` (16 Bits) verbunden. Wenn Sie eine *long int*-Konstante (32 Bits) angeben wollen, müssen Sie hinter der Zahl ein `l` oder `L` angeben, zum Beispiel `123456L`.

Eine Fließkommazahl kann einen Dezimalpunkt enthalten und sowohl in Dezimal- als auch in Exponentialschreibweise angegeben werden, wie in

```
1.234      4.5e+11
```

Escape-Sequenzen in C

Ein String ist eine Folge von Zeichen, die zwischen doppelten Anführungszeichen (") stehen. Man kann auch die C-Konvention mit dem Backslash-Zeichen (\) verwenden, um Spezialzeichen oder einen Buchstaben in einer anderen Notation einzugeben.

Sequenz	Wert	Zeichen
\ \	0x05	Backslash
\ a	0x07	BEL (Alarm)
\ b	0x08	BS (Backspace)
\ f	0x0C	FF (Seitenvorschub)
\ n	0x0A	LF (Zeilenvorschub)
\ r	0x0D	CR (Wagenrücklauf)
\ t	0x09	TAB (Tabulator)
\ v	0x0B	VT (Vertikaler Tabulator)
\ xnnn	nn	Hexadezimal-Byte
\ nnn	nnn	Oktal-Byte

Falls Sie hinter dem Backslash ein anderes Zeichen angeben, das nicht in dieser Liste steht, wird das Zeichen unverändert übernommen.

C-Operatoren und deren Rangfolge

TDW verwendet dieselben Operatoren wie C in derselben Rangfolge. Der Debugger kennt einen neuen Operator, den es in C nicht gibt: den doppelten Doppelpunkt (`::`). Dieser Operator hat eine höhere Priorität als alle Operatoren in C und wird für die Darstellung von FAR-Adressen verwendet, wie zum Beispiel in:

```
0x1234::0x1000

_ES::_BX
```

Die primären Operatoren:

```
[]    .    ->    sizeof
```

haben die höchste, in der Darstellung von links nach rechts fallende Rangfolge. Die einstelligen Operatoren:

```
*     &     -     !     ~     ++     --
```

haben eine geringere, ebenfalls von links nach rechts fallende Priorität, liegen aber noch vor den binären Operatoren. Die binären Operatoren wurden nach fallender Priorität angeordnet, wobei die Operatoren in der gleichen Zeile dieselbe Rangfolge haben:

```
höchste*/%
                    +
                    >><<
                    <><=>=
                    ==!=
                    &
```

```
                    ^
                    |
                    &&
niedrigste||
```

Der einzige Operator mit drei Operanden, `?:`, hat die niedrigste Priorität von allen Operatoren.

Die Zuweisungsoperatoren haben alle dieselbe Priorität und kommen in der Rangfolge noch hinter dem `?:`-Operator:

```
=   +=   -=   *=   /=   %=   >>=   <<=   &=   ^=   |=
```

Die Ausführung von C-Funktionen in Ihrem Programm

In einem C-Ausdruck können Sie Funktionen genauso aufrufen wie in Ihrem Quellcode. TDW führt Ihren Programmcode tatsächlich mit den von Ihnen angegebenen Argumenten aus. Das kann eine sehr praktische Methode sein, um das Verhalten einer von Ihnen geschriebenen Funktion schnell auszutesten. Sie können den Aufruf mit verschiedenen Argumenten wiederholen und dann überprüfen, ob das Ergebnis jedesmal richtig war. Wenn Ihr Programm die folgende Potenzfunktion für x^y enthält,

```
long potenz (int x, int y)
    {
    long  temp = 1 ;
    while (y--)
        temp*= x ;
    return temp ;
    }
```

dann erhält man die in der folgenden Tabelle dargestellten Werte, wenn man die entsprechenden Ausdrücke zur Auswertung angibt:

C-Ausdruck	Ergebnis
potenz(3,2)*2	18
25 +potenz(5,8)	390650
potenz(2)	Fehler (fehlendes Argument)

C-Ausdrücke mit Seiteneffekten

Ein Seiteneffekt tritt dann auf, wenn Sie einen C-Ausdruck auswerten, der den Wert eines Datenobjekts während der Auswertung verändert. In manchen

Fällen sind Seiteneffekte beabsichtigt, um den Wert einer Programmvariablen zu verändern. Manchmal sollen sie aber unbedingt vermieden werden, so daß es wichtig ist zu wissen, wann es zu Seiteneffekten kommen kann.

Die Zuweisungsoperatoren (=, += usw.) verändern den Wert einer Größe, die auf der linken Seite des Operators steht. Die Inkrement- und Dekrementoperatoren (++ und --) verändern den Wert der Größe, die vor oder hinter dem Operator steht, je nachdem, ob der Operator als Präfix- oder Postfixoperator verwendet wird.

Ein raffinierter Seiteneffekt tritt dann auf, wenn man eine Funktion ausführt, die zum Programm gehört. Wenn Sie zum Beispiel den C-Ausdruck

```
funk(1,2,3) + 7
```

auswerten, wird sich Ihr Programm später falsch verhalten, falls `funk` den Wert einer anderen Variablen in Ihrem Programm verändert hat.

Reservierte Wörter in C und die explizite Datentypvorgabe

Mit TDW können Sie die Datentypen für Zeiger genauso vorgeben wie in einem C-Programm. Eine Datentypvorgabe ist die Deklaration eines Datentyps in Klammern. Sie muß vor einem Ausdruck stehen, der einen Zeiger ergibt.

Datentypvorgaben sind sehr praktisch, wenn man den Inhalt einer Speicherstelle untersuchen will, die mit einem `FAR`-Zeiger adressiert wird, der den Operator mit dem doppelten Doppelpunkt (`::`) verwendet. Zum Beispiel:

```
(long far *) 0x3456::0
(char far *)_ES::_BX
```

Man kann eine Datentypvorgabe auch für Programmvariablen verwenden, für die kein Datentyp angegeben wurde. Das ist dann der Fall, wenn ein Modul ohne Debug-Informationen compiliert wurde. Anstatt das Modul neu zu compilieren und zu linken, kann man auch einfach einen Datentyp vor dem Variablennamen angeben, falls man den Datentyp der Variablen kennt. Wenn die Variable `iptr` zum Beispiel ein Zeiger auf einen Integer-Wert ist, kann man mit folgendem C-Ausdruck

```
* (int *) iptr
```

auf den Integer-Wert zugreifen.

Sie können zu diesem Zweck ebenso den Befehl *Type cast* im lokalen Menü des *Inspector*-Fensters benutzen.

TDW stellt zwei reservierte Bezeichnungen, nämlich `lh2fp` und `gh2fp`, für die Dereferenzierung von in Microsoft Windows verwendeten Memory Handles bereit.

Die folgenden reservierten Bezeichnungen können Sie zur Datenumwandlung in TDW angeben:

```
char     float    near      unsigned
double   huge     short
```

```
enum     int      struct
far      long     union
```

Assembler-Ausdrücke

TDW unterstützt die gesamte Syntax der Assembler-Ausdrücke. Ein Ausdruck besteht aus einer Kombination von Operatoren, Strings, Variablen und Konstanten. All diese Bestandteile eines Ausdrucks werden im folgenden Abschnitt eingehend besprochen.

Symbole in Assembler

Ein Symbol ist eine vom Benutzer definierte Bezeichnung für ein Datenobjekt oder eine Routine in einem Programm. Ein Assembler-Symbol kann mit einem Buchstaben (`a-z`, `A-Z`) oder einem der vier Sonderzeichen `@`, `?`, `_` und `$` anfangen. Die nachfolgenden Zeichen im Symbolnamen können sich aus Ziffern (`0-9`), Buchstaben und den bereits erwähnten Sonderzeichen zusammensetzen. Den Punkt (`.`) kann man ebenfalls als erstes Zeichen in einem Symbol verwenden, aber nicht innerhalb des Namens.

Das Spezialsymbol `$` bezieht sich auf die aktuelle Position im Programm, die vom Registerpaar CS:IP angezeigt wird.

Assembler-Konstanten

Als Konstanten sind entweder ganze Zahlen oder Fließkommazahlen zulässig. Eine Fließkommazahl kann einen Dezimalpunkt enthalten. Es kann sowohl die Dezimal- als auch die Exponentialschreibweise verwendet werden, wie zum Beispiel in:

```
1.234   4.5e+11
```

Eine Integer-Konstante wird hexadezimal interpretiert, wenn nicht eines der folgenden Formate vorgegeben ist:

Format	Zahlenbasis
*Ziffern*H	hexadezimal
*Ziffern*O	oktal
*Ziffern*Q	oktal
*Ziffern*D	dezimal
*Ziffern*B	binär

Eine Hexadezimalzahl muß immer mit einer der Ziffern von `0` bis `9` beginnen. Wenn Sie eine Zahl eingeben wollen, die mit einem der Buchstaben von `A-F` anfängt, müssen Sie eine Null davorschreiben. Wenn eine Hexadezimalzahl auf `D` oder `B` enden soll, müssen Sie die Zahl mit `H` abschließen.

Operatoren in Assembler

TDW unterstützt die meisten Assembler-Operatoren; die folgende Tabelle enthält diese Operatoren in der Reihenfolge ihrer Priorität:

```
xxx PTR (BYTE PTR...)
. (Auswahl eines Strukturelements)
: (Segmentvorgabe)
OR XOR
AND
NOT
EQ NE LT LE GT GE
+ -
* / MOD SHR SHL
+ (einstellig) - (einstellig)
OFFSET SEG
() []
```

Die Variablen können mit dem Zuweisungsoperator (`=`) geändert werden, wie zum Beispiel in:

```
a = [BYTE PTR DS:4]
```

Die Formatsteuerung

Wenn Sie einen Ausdruck angeben, dessen Ergebnis dargestellt werden soll, gibt TDW das Ergebnis in einem Format aus, das vom jeweiligen Datentyp im Ausdruck abhängt. Wollen Sie das Format für die Darstellung eines Ausdrucks verändern, können Sie ein Komma hinter dem Ausdruck angeben und eine Formatangabe hinter das Komma schreiben. Die Formatangabe besteht aus einem Wiederholungsfaktor und einem Formatbuchstaben. Wiederholungsfaktoren dürfen nur bei Zeigern oder Arrays angegeben werden. Beachten Sie bitte, daß eine Formatangabe mit dem falschen Datentyp nicht beachtet wird.

Buchstabe	Format
c	stellt ein Zeichen oder einen Stringausdruck in jedem Fall als Zeichen dar. Nomalerweise werden alle Zeichen, die nicht ausgegeben werden können, als EscapeSequenz oder

Buchstabe	Format
	numerisch dargestellt. Mit dieser Option werden die Zeichen mit Hilfe des vollständigen IBMZeichensatzes ausgegeben.
d	stellt eine ganze Zahl als Dezimalzahl dar.
f[#]	stellt eine Fließkommazahl mit der angegebenen Anzahl von Ziffern dar. Wenn Sie die Anzahl der Ziffern nicht angeben, dann werden so viele Ziffern verwendet, wie benötigt werden.
m	stellt einen Speicherausdruck mit Hexadezimal-Bytes dar.
md	stellt einen Speicherausdruck mit Dezimal-Bytes dar.
p	stellt den Wert als Zeiger dar, wobei Segmente wenn möglich mit einer Registerbezeichnung angegeben werden. Es wird auch das Datenobjekt, auf das der Zeiger verweist, ausgegeben. Dies ist die Voreinstellung, wenn keine Formatangabe vorliegt.
s	stellt ein Array oder einen Zeiger auf ein Zeichen-Array als String in Anführungszeichen dar. Der String muß mit NUL abgeschlossen sein.
x oder h	stellt eine ganze Zahl als Hexadezimalzahl dar.

10. Objektorientiertes Debuggen

Turbo Debugger wurde erweitert, damit die Anforderungen der Neuerungen objektorientierter Programmierung in C++ für Windows voll erfüllt werden. Außer den Erweiterungen für die schrittweise Bearbeitung der Elementfunktionen einer Klasse und die Untersuchung von Objekten oder Klassen im Dialogfenster *Evaluate/Modify* und dem *Watches* -Fenster weist TDW eine Reihe neuer Spezialfenster und lokaler Menüs auf, die speziell für Objekte und Klassen entwickelt wurden.

Das Fenster Hierarchy

TDW hat ein spezielles Fenster für die Untersuchung von Objekt- oder Klassenhierarchien. Dieses *Hierarchy*-Fenster erhalten Sie mit dem Befehl *View/Hierarchy*.

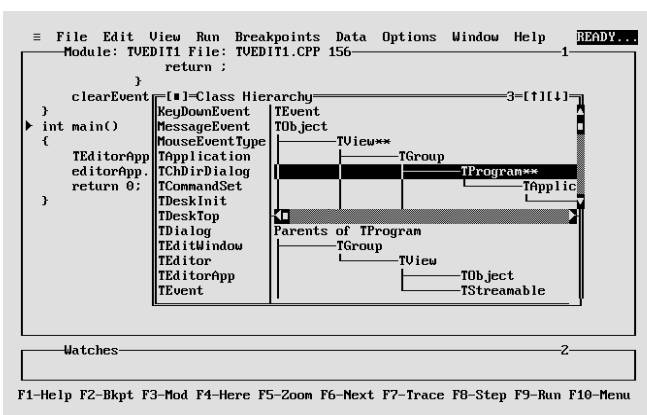


Abbildung 10.1 Das Fenster Hierarchy

Das *Hierarchy*-Fenster stellt Informationen über die *Typen* von Objekten oder Klassen dar, und nicht über die Instanzen. Das Fenster besteht aus entweder

zwei oder drei Ausschnitten. Der dritte Ausschnitt erscheint jedoch nur bei C++ Programmen die mit Mehrfachvererbung arbeiten.

- Der linke Ausschnitt (Klassenliste) enthält die einzelnen Typen von Objekten oder Klassen, die im untersuchten Modul vorkommen.

- Der rechte Ausschnitt (der bei einer Mehrfachvererbung in C++für Windows in zwei weitere Ausschnitte unterteilt ist) zeigt alle Objekte und Klassen in Ihren Hierarchien, wobei eine Liniengrafik verwendet wird, in der am linken Rand des Ausschnitts die Ausgangsobjekte erscheinen und die davon abstammenden Objekte rechts und unterhalb davon dargestellt werden. Die Verbindungslinien zeigen die Beziehungen zwischen Vorfahren und Nachfahren. Jede Klasse, die durch einen Stern (*) gekennzeichnet ist, bedeutet Mehrfachvererbung verschiedener Basisklassen und diese stehen im unteren Ausschnitt.

- Der rechte untere Auschnitt (Vorfahrenbaum) enthält, soweit er erscheint, alle Basisklassen mit Mehrfachvererbung.

Die Listenausschnitt mit den Objekttypen

Der linke Ausschnitt zeigt eine alphabetische Liste aller Objekt- oder Klassentypen, die im aktuellen Modul verwendet werden. Sie können nun mit den Pfeiltasten den Auswahlbalken auf den Objekttyp positionieren, den Sie näher untersuchen wollen. In einer sehr langen Liste von Objekttypen ist es aber einfacher, wenn Sie eine andere Vorgehensweise wählen: Beginnen Sie den Namen des gewünschten Objekttyps einzugeben. Sie werden sehen, daß der Auswahlbalken auf den Objekttyp springt, dessen Namen mit der Zeichennolge beginnt, die Sie bisher eingegeben haben. Dies bezeichnet man auch als sukzessive Auswahl. Drücken Sie die Taste [CR], um ein Objekttyp-/Klassen-*Inspector*-Fenster für das ausgewählte Objekt zu öffnen. Die Objekttyp-/Klassen-*Inspector*-Fenster werden nachfolgend noch ausführlich besprochen.

Das lokale Menü der Objekttyp-/Klassenliste

Drücken Sie [Alt][F10], um das lokale Menü dieses Ausschnitts anzuzeigen. Sie können auch die [Strg]-Tastenkürzel verwenden, wenn diese in TDWINST aktiviert wurden. Dieses lokale Menü enthält zwei Einträge:

Inspect öffnet ein Objekttyp-Inspector-Fenster für die ausgewählte Klasse oder den Objekttyp.

Tree springt in den rechten Ausschnitt des Fensters, in dem der Hierarchiebaum dargestellt ist und positioniert den Auswahlbalken auf den im linken Ausschnitt ausgewählten Objekt-/Klassentyp.

Der Ausschnitt mit dem Objektbaum

Der rechte Ausschnitt stellt den Hierarchiebaum aller Objekte oder Klassen dar, die im aktuellen Modul verwendet werden. Die Beziehungen zwischen Vorfahren und Nachkommen werden durch Verbindungslinien angezeigt, wobei die Nachkommen rechts und unterhalb der Vorfahren stehen.

Um ein einzelnes Objekt in einem komplexen Hierarchiebaum zu finden, gehen Sie in den linken Ausschnitt und verwenden die oben beschriebene sukzessive Auswahl. Anschließend wählen Sie den Befehl *Tree* im lokalen Menü aus, um wieder in den Hierarchiebaum zurückzukehren. Der Auswahlbalken befindet sich dann über dem ausgewählten Objekt.

Wenn Sie CR eingeben, erscheint ein Objekttyp-/Klassen-Inspector-Fenster für das ausgewählte Objekt.

Das lokale Menü des Hierarchiebaums

Das lokale Menü des Hierarchiebaums (Alt F10 im rechten Ausschnitt) hat nur einen Eintrag: *Inspect*. Wenn Sie diesen Befehl auswählen, erscheint ein Objekttyp-/Klassen-*Inspector*-Fenster für den ausgewählten Objekttyp. Schneller geht es jedoch, indem Sie einfach CR eingeben, wenn Sie den ausgewählten Objekttyp untersuchen wollen.

Falls Sie ein Programm geladen haben, das in C++ verfaßt wurde und Klassen mit Mehrfachvererbung aufweist, enthält das lokale Menü des Hierarchie-Ausschnitts einen weiteren Befehl, *Parents*. Damit wird die Anzeige der Vorfahren einer Klasse im Vorfahrenausschnitt umgeschaltet. Die Voreinstellung für *Parents* ist *Yes*.

Der Ausschnitt mit dem Vorfahrenbaum

Wenn Sie ein Programm in C++ für Windows geladen haben, in dem Klassen mit Mehrfachvererbung vorkommen, erscheint unterhalb des Hierarchieausschnitts ein weiterer Ausschnitt, der Vorfahrenbaum im Fenster *Hierarchy*. Falls die untersuchte Klasse mehrere Vorfahren hat und der Befehl *Parent* des lokalen Menüs im Hierarchie-Ausschnitt auf *Yes* eingestellt ist, erscheint ein auf dem Kopf stehender Baum im Vorfahrenausschnitt mit der Meldung `Parents of Class` (engl. Vorfahren der Klasse) am linken Rand des Ausschnitts. Die Vorfahren werden rechts und unterhalb davon dargestellt, wobei die Beziehungen zwischen Vorfahren und Nachkommen durch Verbindungslinien angedeutet werden.

Sie können wie im Hierarchiebaum ein Objekttyp-/Klassen-*Inspector*-Fenster für jede Klasse öffnen, die im Vorfahrenbaum erscheint.

Das lokale Menü des Vorfahrenbaums

Der Ausschnitt mit dem Vorfahrenbaum hat schließlich auch noch ein eigenes Menü mit einem einzigen Befehl: *Inspect*. Dieser funktioniert genauso

wie der *Inspect*-Befehl im lokalen Menü des Hierarchie-Ausschnitts: Es wird ein *Inspector*-Fenster für den ausgewählten Typ von Objekt oder Klasse geöffnet.

Das Inspector-Fenster für Objekttypen/Klassen

TDW hat eine neue Art von *Inspector*-Fenster, mit dem man die Einzelheiten eines Objekttyps oder einer Klasse untersuchen kann: das *Inspector*-Fenster für Objekttypen und Klassen. Dieses Fenster faßt alle Informationen über die Datentypen des Objekts zusammen, bezieht sich aber nicht auf eine bestimmte Instanz eines Objekts.

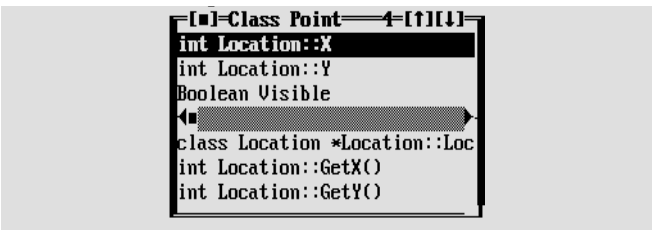


Abbildung 10.2
Das Inspector-Fenster für Objekttypen/Klassen

Das Fenster ist waagerecht in zwei Ausschnitte unterteilt. Der obere Ausschnitt enthält die Datenfelder des Objekttyps und der untere Ausschnitt die Bezeichnungen der Elementfunktionen. Mit der Tab-Taste können Sie sich zwischen den beiden Ausschnitten dieses *Inspector*-Fensters hin- und herbewegen.

Wenn das ausgewählte Datenfeld ein Objekttyp oder ein Zeiger auf einen Objekttyp ist, können Sie mit CR ein weiteres Objekttyp-/Klassen-Inspector-Fenster für den ausgewählten Typ öffnen. (Diese Aktion ist mit der Auswahl des Befehls *Inspect* im lokalen Menü dieses Fensters identisch.) Auf diese Weise lassen sich auch kompliziert verschachtelte Strukturen von Objekten mit minimalem Tippaufwand schnell untersuchen.

Damit die Darstellung übersichtlich bleibt, werden die Parameter der Elementfunktionen im Objekttyp-/Klassen-*Inspector*-Fenster nicht gezeigt. Um die Parameter zu untersuchen, wählen Sie die Elementfunktion mit CR aus. Es erscheint ein *Inspector*-Fenster zur Untersuchung von Methoden und Elementfunktionen. Der obere Ausschnitt des Fensters enthält die Adresse der Implementierung der Klasse sowie die Bezeichner und Datentypen aller Parameter.

Drückt man irgendwo innerhalb dieses Inspector-Fensters CR, so erscheint das *Module*-Fenster oder das *CPU*-Fenster im Vordergrund mit dem Cursor an der Stelle des Quelltexts, an der die untersuchte Elementfunktion implementiert ist.

Wie alle *Inspector*-Fenster läßt sich auch hier das aktuelle *Inspector*-Fenster mit Esc schließen; mit Alt F3 werden alle geschlossen.

Die lokalen Menüs des Inspector-Fensters für Objekttypen/Klassen

Das lokale Menü des oberen Ausschnitts mit den Objektdaten enthält folgende Menüpunkte:

Drückt man Alt F10 , erhält man das lokale Menü des jeweiligen Ausschnitts. Wenn die Strg -Tastenkürzel aktiviert sind (in TDWINST), können Sie einen lokalen Menübefehl mit Strg und dem ersten Buchstaben des Befehls aufrufen.

Der Ausschnitt *Data Member* enthält folgende Menübefehle:

Inspect

Wenn das ausgewählte Feld ein Objekttyp, eine Klasse oder ein Zeiger auf eines dieser Elemente ist, wird ein neues *Inspector*-Fenster zur Darstellung von Objekttypen/Klassen für das ausgewählte Feld geöffnet.

Hierarchy

Der Befehl *Hierarchy* öffnet ein *Hierarchy*-Fenster für den untersuchten Objekttyp oder Klasse. Das *Hierarchy*-Fenster ist bereits zu Beginn des Kapitels beschrieben worden.

Show Inherited

Yes ist die Standardeinstellung für diesen Schalter. Wenn er auf *Yes* steht, werden alle Datenfelder und Methoden angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Felder und Methoden angezeigt, die im Typ des untersuchten Objekts definiert wurden.

Im lokalen Menü des unteren Ausschnitts *Member Function* sind die Befehle *Inspect*, *Hierarchy* und *Show Inherited* enthalten:

Inspect

Für das ausgewählte Element wird ein *Inspector*-Fenster für Elementfunktionen geöffnet. Wenn Sie zu dem Zeitpunkt Strg I drücken, da der Cursor über der Adresse im *Inspector*-Fenster für Elementfunktionen steht, erscheint das *Module*-Fenster mit dem Cursor an der Stelle, an der die untersuchte Elementfunction implementiert ist.

Hierarchy

Dieser Befehl öffnet ein *Hierarchy*-Fenster für den untersuchten Objekt- oder Klassentyp. Das *Hierarchy*-Fenster ist zu Beginn dieses Kapitels bereits beschrieben worden.

Show Inherited

Yes ist die Standardeinstellung dieses Schalter. Wenn er auf *Yes* steht, werden alle Elementfunktionen angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Elementfunktionen angezeigt, die im Typ des untersuchten Objekts definiert wurden.

Inspector-Fenster für Objekt-/Klasseninstanzen

Das *Inspector*-Fenster Objekttypen und Klassen enthält Informationen über Objekttypen, sagt aber nichts darüber aus, welche Daten in einer bestimmten Instanz eines Objekts oder einer Klasse zu einem bestimmten Zeitpunkt der Programmausführung enthalten sind. TDW hat eine erweiterte Form des altbekannten Inspector-Fensters für Records, das speziell auf die Untersuchung von Objekt- und Klasseninstanzen ausgelegt ist. Sie erhalten dieses Fenster, indem Sie den Cursor auf eine Objekt oder Klasseninstanz im *Module*-Fenster stellen und dann Strg I drücken.

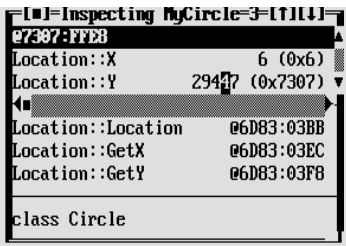


Abbildung 10.3 Das Inspector-Fenster für Objekt-/Klasseninstanzen

Die meisten *Inspector*-Fenster von TDW für Records haben zwei Ausschnitte: einen oberen Ausschnitt, der die Feldbezeichnungen und deren aktuelle Werte zusammenfaßt, und einen unteren Ausschnitt, der den Datentyp des Felds angibt, das im oberen Ausschnitt ausgewählt ist. Auch ein *Inspector*-Fenster für Objekt-/Klasseninstanzen verfügt über diese beiden Ausschnitte — dazwischen ist aber noch ein dritter Ausschnitt dargestellt. Der neue Ausschnitt stellt die Elementfunktionen der Instanz mit ihren Programmadressen dar.

Die lokalen Menüs des Inspector-Fensters für Objekt- und Klasseninstanzen

Jeder der beiden oberen Ausschnitte des *Inspector*-Fensters für Objekt- und Klasseninstanzen hat sein eigenes lokales Menü, das man mit Alt F10 erhält. Auch hier kann man wieder die Strg -Tastenkürzel verwenden (sofern sie in TDWINST aktiviert wurden), um die einzelnen Menübefehle direkt aufzurufen.

Wie bei den *Inspector*-Fenstern von Records dient der untere Ausschnitt nur der Darstellung des Typs im ausgewählten Feld und hat kein lokales Menü.

Der obere Ausschnitt, in dem die Datenfelder oder -elemente zusammengefaßt werden, weist folgende Befehle im lokalen Menü auf:

Range...

Dieser Befehl hat sich gegenüber den früheren Versionen nicht verändert. Hier wird der Bereich für die Elemente eines Arrays angegeben, der angezeigt

werden soll. Wenn das untersuchte Element kein Array oder Zeiger ist, kann man diesen Befehl nicht aufrufen.

Change...
Mit diesem Befehl kann man den Wert des ausgewählten Datenfelds verändern. Dieser Befehl hat sich gegenüber den früheren Versionen von TDW ebenfalls nicht geändert.

Methods
Dieser Befehl ist ein *Yes*/*No*-Schalter mit der Standardeinstellung *Yes*. Falls er auf *Yes* eingestellt ist, werden die Methoden oder Elementfunktionen im mittleren Ausschnitt dargestellt. Wenn er auf *No* steht, erscheint der mittlere Ausschnitt nicht. Die Einstellung wird vom nächsten geöffneten *Inspector*-Fenster übernommen.

Show Inherited
Dieser Menüpunkt funktioniert ebenfalls als *Yes*/*No*-Schalter. Wenn er auf *Yes* steht, werden alle Datenfelder und Methoden angezeigt, ob sie nun innerhalb des Typs des untersuchten Objekts definiert sind oder von einem Vorfahren vererbt wurden. Wenn er auf *No* steht, werden nur die Methoden und Datenfelder angezeigt, die im Typ des untersuchten Objekts definiert sind.

Inspect
Wie bei den früheren Versionen von TDW wird nach der Auswahl dieses Befehls ein *Inspector*-Fenster für die Daten des ausgewählten Felds geöffnet. Mit der Taste CR erreicht man dasselbe, wenn sich der Auswahlbalken über dem ausgewählten Feld oder Element befindet.

Descend
Das ausgewählte Element tritt an die Stelle des aktuellen *Inspector*-Fensters. Es wird kein neues *Inspector*-Fenster geöffnet. Sie können aber nicht mehr zum vorher untersuchten Feld zurückkehren, wie das bei der Option *Inspect* möglich ist. *Descend* empfiehlt sich für die Untersuchung komplexer Datenstrukturen, wie Listen, da man für die einzelnen Elemente der Datenstruktur kein eigenes *Inspector*-Fenster öffnen muß. (Der Befehl *Descend* empfiehlt sich vor allem zur Untersuchung komplexer Datenstrukturen, wenn kein eigenes *Inspector*-Fenster geöffnet werden soll.)

New Expression...
Dieser Befehl fragt nach einem neuen Datenelement oder Ausdruck, der untersucht werden soll. Das neue Element ersetzt den aktuellen Ausdruck oder das aktuelle Element im Fenster. Es wird kein zusätzliches Fenster geöffnet.

Type cast...
Mit diesem Befehl können Sie einen anderen Datentyp (`byte`, `word`, `int`, `char`, Zeiger) für das untersuchte Element vorgeben. Dieser Befehl ist vor allem dann sehr nützlich, wenn im *Inspector*-Fenster ein Symbol vorkommt, für das keine

Datentypinformationen vorliegen, oder Zeiger untersucht werden soll, für den kein expliziter Typ angegeben wurde.

Hierarchy
Nach Auswahl dieses Befehls wird ein *Hierarchy*-Fenster geöffnet. Eine ausführliche Beschreibung dieses Fensters finden Sie am Anfang dieses Kapitels.

Der mittlere und der untere Ausschnitt

Im mittleren Ausschnitt werden die die Elementfunktionen einer Klasse zusammengefaßt. Der einzige Unterschied zwischen den lokalen Menüs des Ausschnitts der Elementfunktionen und des Datenausschnitts ist das Fehlen des *Change*-Befehls. Im Gegensatz zu Datenfeldern und Datenelementen darf man die Adressen von Elementfunktionen während der Ausführung eines Programms nicht verändern.

Im unteren Ausschnitt wird der Datentyp des Elements dargestellt, das in den beiden oberen Fenstern ausgewählt ist.

11. Debuggen unter Windows

Dieses Kapitel befaßt sich mit den Fähigkeiten von TDW, auf Windows-spezifische Informationen zuzugreifen, zum Beispiel:

- Protokollieren von Botschaften, die in den Fenstern Ihrer Anwendung gesendet oder empfangen werden;
- den globalen Heap;
- den lokalen Heap;
- Erstellung einer kompletten Liste der Module, die von Windows geladen werden, einschließlich der dynamischen Link-Bibliotheken (DLLs);
- das Debuggen von dynamischen Link-Bibliotheken (DLLs);
- Selektoren im Protected-Mode (im CPU-Fenster).

Spezialitäten für Windows

Die folgenden Eigenschaften von TDW unterstützen das Debuggen von Windows-Anwendungen:

- Ein *View*-Fenster, das Fenster *Windows Messages* , das die Botschaften zeigt, die an die Fenster Ihres Programms weitergegeben werden;
- Sie können drei Arten von Daten im *Log*-Fenster ausgeben lassen:
  - Die Datensegmente im lokalen Heap Ihres Programms;
  - die Datensegmente des globalen Heap;
  - eine komplette Liste der Module Ihres Programms einschlließlich aller dynamischer Link-Bibliotheken (DLLs);
- Konvertierung von Ausdrücken (Speicher-Handles in FAR-Zeiger);
- Unterstützung beim Debuggen von DLLs im Fenster Load Module Source oder im Fenster DLL Symbols (wählen Sie hierzu den Befehl *View/Modules*);

■ der Selector-Ausschnitt des CPU-Fensters; diesem Ausschnitt können Sie die Inhalte beliebiger Selektoren im Protected-Mode entnehmen.

Protokollierung von Windows-Botschaften

Zur Verfolgung von Botschaften, die an Fenster Ihres Programms geschickt werden, wählen Sie den Befehl *View/Windows Messages* , mit dem das Windows-Messages-Fenster geöffnet wird. In diesem Fenster werden die Botschaften angezeigt, die Windows an eines oder mehrere Fenster in Ihrem Programm überträgt.

Das Fenster *Windows Messages* besteht aus drei Ausschnitten, dem Ausschnitt *Window Selection* (oben links), dem Ausschnitt *Message Class* (oben rechts) und dem Ausschnitt *Messages* (unten). Die Botschaften werden im Ausschnitt *Messages* angezeigt.

Der Aufbau dieses Fensters und die Art und Weise, wie Sie die Fenster Ihres Anwendungsprogramms dort eintragen können, hängt davon ab, ob Sie ein ObjectWindows- oder Standard-Windows-Programm debuggen.

Auswahl eines Fensters für Standard-Windows-Programme

Wenn Sie beim Debuggen eines Standard-Windows-Programms *View/ Windows Messages* auswählen, erscheint folgendes Fenster:

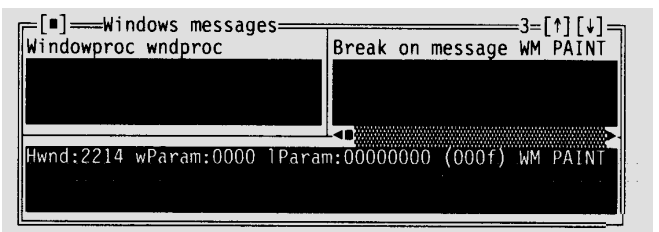


Abbildung 11.1 Das Fenster Windows Messages in Standard-Windows-Anwendungen

Bevor Sie Botschaften protokollieren können, müssen Sie zuerst angeben, für welches Fenster Sie die Botschaften protokollieren wollen. Dies geschieht im Ausschnitt oben links, dem Ausschnitt *Window Selection* . Über das lokale Menü dieses Ausschnitts (Aufruf durch Alt F10) können Sie eine Fensterauswahl hinzufügen, löschen oder alle ausgewählten Fenster löschen.

Hinzufügen einer Fensterauswahl Um eine Fensterauswahl hinzuzufügen, wählen Sie entweder *Add* aus dem lokalen Menü des Ausschnitts *Window Selection* oder Sie schreiben in den Ausschnitt. Beide Methoden öffnen das Dialogfenster *Add Window* , dargestellt auf der folgenden Seite.

Sie können den Namen der Routine, die die Botschaften für dieses Fenster verarbeitet, eingeben (wählen Sie *Window Proc*) oder einen Handlewert (wählen Sie *Handle*). Geben Sie soviele Routinen oder Handles an wie nötig sind, um die Botschaften für Ihre Fenster zu verfolgen. (Durch Eingabe des ersten Wertes in diesen Ausschnitt wird die Botschaftenklasse auf *Log all messages* gesetzt.)

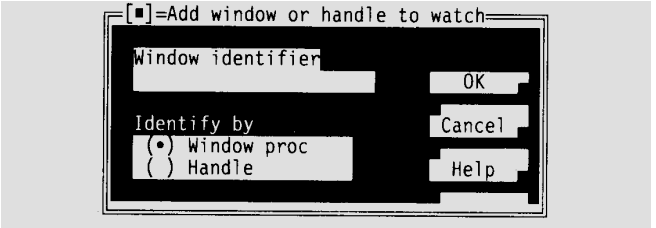


Abbildung 11.2 Das Dialogfenster Add Window für Standard-Windows-Anwendungen

Es ist einfacher, ein Fenster über den Namen der Routine, die seine Botschaften verarbeitet, anzugeben (z.B. WndProc), weil Sie den Namen einer Routine jederzeit nach dem Laden Ihres Programms eingeben können.

Wenn Sie lieber eine Handle-Variable eingeben möchten, müssen Sie zuerst im Programm hinter die Zeile gehen, in der der Variablen ein Handle zugewiesen wird. (Verwenden Sie die Taste F7 oder F8 um das Programm schrittweise abzuarbeiten.) Wenn Sie den Variablennamen eingeben, ohne hinter die Zuordnungsanweisung zu gehen, kann TDW keine Botschaften dafür aufnehmen.

Die Auswahl eines Fensters für ein ObjectWindows-Programm

Wenn Sie beim Debuggen eines ObjectWindows-Programms *View/Windows Messages* auswählen, erhalten Sie die Standardausführung des Dialogfensters *Windows Messages* , wie sie in Abbildung 11.1 dargestellt wurde. Dieses Dialogfenster funktioniert für ObjectWindows-Programme genauso wie für Standard-Windows-Programme, aber Sie können keine Bezeichnungen von Windows-Prozeduren verwenden. Stattdessen müssen Sie das Handle für das Fensterobjekt angeben, dessen Botschaften Sie protokollieren oder mit einem Breakpoint versehen wollen.

Ermitteln des Handles eines Fensters

Bevor Sie das Handle eines Fensterobjekts verwenden könnnen, müssen Sie Ihr Programm bis hinter die Stelle ausführen, an der das Handle initialisiert wird. Dafür gibt es eine Reihe von Möglichkeiten.

- Am einfachsten ist es, wenn Sie Ihr Programm starten und mit Strg Alt S-Abf zu TDW zurückkehren.

- Eine weitere Möglichkeit ist das Setzen eines Breakpoints in einer Ihrer Programmroutinen, in denen Botschaften verarbeitet werden (zum Beispiel in einer Routine, die WM_MOUSEMOVE-Botschaften verarbeitet). Anschließend starten Sie das Programm und lösen den Breakpoint für das jeweilige Fenster aus (indem Sie zum Beispiel die Maus bewegen).

- Wenn Sie mit dem Fensterobjekt größere Probleme haben – zum Beispiel einen UAE-Fehler (Unrecoverable Application Error) bei der ersten Anzeige des Fensters – müssen Sie das Handle des Fensters über einen Umweg ermitteln. Da das Fenster über die ObjectWindows-Funktion *CreateWindows* initialisiert wird und diese Funktion ausgeführt wird, nachdem Sie das Fenster initialisieren, müssen Sie diese Funktion zum Ermitteln des Handles in Ihrer Fensterklasse überschreiben und dort einen Breakpoint auf die Funktion setzen.

Im folgenden Programmauschnitt wird diese Funktion für die Fensterklasse *ScribbleWindow* im Programm TDODEMO überschrieben:

```
void ScribbleWindow::SetupWindow()
{
    TWindow::SetupWindow();
}
```

Anschließend stellen Sie den Cursor in die Zeile nach der Initialisierungsanweisung und drücken F4, um das Programm bis zu der Stelle auszuführen, an der das Handle des Fensters, Dialogfensters oder Bildschirmelements initialisiert wird. In diesem Fall würden Sie den Cursor in die Zeile mit der abschließenden geschweiften Klammer der Funktion *SetupWindow* stellen.

Wenn das Handle initialisiert und das Programm zu TDW zurückgekehrt ist, können Sie den Wert mit *Data/Inspect* ermitteln und den Namen des damit verbundenen Fensterobjekts eingeben (in TDODEMO wäre dies *WinMain#MyApp.MainWindow*). Suchen Sie jetzt nach dem Datenelement *HWindow*, und kopieren Sie es (mit Umschalt F3) in die Zwischenablage. Sie können den Inhalt von *HWindow* (mit Umschalt F4) aus der Zwischenablage in das Dialogfenster *Add* des linken oberen Fensterausschnitts von *Windows Messages* kopieren.

Auswahl eines Fenster mit ObjetWindows-Unterstützung

Wenn Sie das Programm TDWINST mit der Option *-w* starten, können Sie die Unterstützung von ObjectWindows-Botschaften in TDW aktivieren. Mit Hilfe dieser Option können Sie die Namen der Fensterobjekte so verwenden, wie Sie diese in Ihrem Programm deklariert haben.

Wählen Sie bei aktivierter OWL-Option (ObjectWindows Library) den Befehl *View/Window Messages* aus, wird das Fenster wie folgt angezeigt:

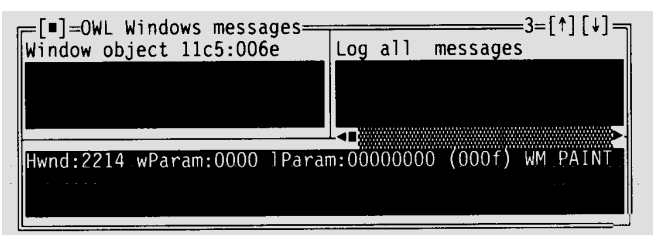


Abbildung 11.3 Das Fenster Windows Messages bei ObjectWindows-Unterstützung

Bevor Sie Botschaften protokollieren können, müssen Sie zuerst angeben, für welches Fenster, Dialogfenster oder Bildschirmelement die Botschaften aufgezeichnet werden sollen. Dies tun sie im linken oberen Ausschnitt, dem Ausschnitt *Windows Selection*. Mit dem lokalen Menü (das Sie mit Alt F10 aktivieren) können Sie ein Fensterobjekt einfügen, löschen oder alle Fensterobjekte auf einmal löschen.

Hinzufügen einer Fensterauswahl mit ObjectWindows-Unterstützung

Bevor Sie ein Fensterobjekt verwenden könnnen, müssen Sie Ihr Programm bis hinter die Stelle ausführen, an der das Handle initialisiert wird. Normaler-

weise wird das Objekt in einer Anweisung initialisiert, zum Beispiel wie in der folgenden Funktionsdefinition von TDODEMO:

```
void CScribbleApplication::InitMainWindow()
{
    MainWindow::new ScribbleWindow(NULL, Name);
}
```

Stellen Sie den Cursor in die Zeile nach der Initialisierungsanweisung und drücken F4, um das Programm bis zu der Stelle auszuführen, an der das Fenster, Dialogfenster oder Bildschirmelement initialisiert wird. In diesem Fall würden Sie den Cursor in die Zeile mit der abschließenden geschweiften Klammer der Funktion *ScribbleWindow* stellen.

Wenn das Fensterobjekt initialisiert ist, können Sie es in den Ausschnitt *Windows Selection* einfügen. Zum Einfügen des Objekts wählen Sie entweder den Befehl *Add* im lokalen Menü von *Windows Selection* aus oder geben die Anfangsbuchstaben des Objekts in den Ausschnitt ein. Mit beiden Möglichkeiten erhalten Sie das Dialogfenster *Add Window*.

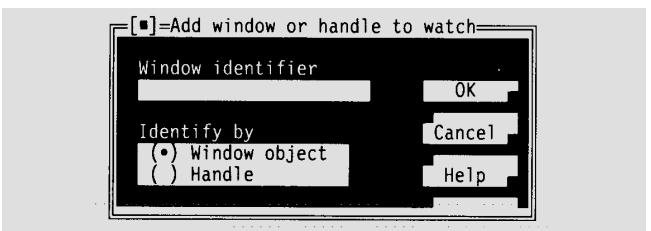


Abbildung 11.4 Das Dialogfenster Add Window bei ObjectWindows-Unterstützung

Wenn Sie nicht genaus wissen, wo das Objekt deklariert wurde, müssen Sie den Gültigkeitsbereich vorgeben, um auf das Objekt zugreifen zu können. In TDODEMO ist *MainWindow* zum Beispiel ein Element von *MyApp* (da *MyApp* vom Typ *CScribbleApplication* ist, das wiederum von *TApplication* abgeleitet wurde, zu dem das Datenelement *MainWindow* gehört). Da *MyApp* aber in der Funktion *WinMain* deklariert ist, können Sie nicht auf *MyApp* zugreifen, falls sich das Programm nicht zufälligerweise in *WinMain* befindet. Falls Sie sich im selben Modul befinden, lautet die Eingabe mit vorgegebenen Gültigkeitsbereich:

```
WinMain#MyApp.MainWindow
```

Sie können entweder den Namen der Routine angeben, in der die für dieses Fenster, Dialogfenster oder Bildschirmelement bestimmten Botschaften verarbeitet werden (wählen Sie dazu den Schalter *Window Object*) oder den Wert für das entsprechende Handle (wählen Sie dazu den Schalter *Handle*). Geben Sie alle benötigten Objektnamen oder Handle-Werte ein, deren Botschaften Sie in Ihren Fenstern untersuchen wollen. (Beim Einfügen des ersten Objekts in diesem Ausschnitt wird die Botschaftenklasse automatisch auf *Log all messages* gesetzt.)

Löschen einer Fensterauswahl Zum Löschen einer Fensterauswahl plazieren Sie den Cursor auf den gewünschten Eintrag. Dann wählen Sie

entweder *Remove* aus dem lokalen Menü oder Sie drücken Entf, Strg Y oder Strg R.

Mit *Delete All* im lokalen Menü löschen Sie alle ausgewählten Fenster.

Eingabe von Botschaftenklasse und Aktion

Der Ausschnitt rechts oben ist der Ausschnitt *Message Class*. Über sein lokales Menü, das identisch mit dem lokalen Menü des Ausschnitts *Window Selection* ist, können Sie eine Botschaftenklasse hinzufügen, löschen oder alle von Ihnen hinzugefügten Klassen löschen.

Bevor Sie eine Botschaftenklasse in diesem Ausschnitt hinzufügen können, müssen Sie eine Fensterprozedur oder ein Handle im Ausschnitt *Window Selection* angeben.

Wenn Sie keine zu überwachende Botschaft oder Botschaftenklasse spezifizieren, überwacht TDW alle Botschaften, die an die Fensterprozedur oder das Handle übertragen werden.

Hinzufügen einer Botschaftenklasse Durch Anwahl von *Add* im lokalen Menü des Ausschnitts *Message Class* fügen Sie eine Botschaftenklasse hinzu.

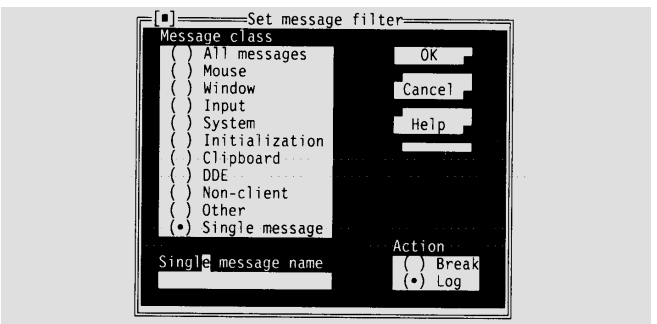


Abbildung 11.5 Das Dialogfenster Set Message Filter

Das Dialogfenster *Set Message Filter* öffnet sich und verlangt die Eingabe der zu verfolgenden Botschaftenklasse und der Aktion, die beim Empfang dieser Botschaft ausgeführt werden soll.

Standardmäßig protokolliert TDW alle Botschaften die mit WM_ beginnen. Da jedoch so viele Botschaften eingehen, möchten Sie wahrscheinlich die Bandbreite durch Auswahl einer Botschaftenklasse aus der Botschaftenklassenliste begrenzen. Da Sie immer nur eine Klasse angeben können, müssen Sie zur Verfolgung mehrerer Botschaftenklassen für jede Klasse, die Sie setzen wollen, die Option *Add* benützen.

In der folgenden Tabelle werden die verschiedenen Botschaftenklassen beschrieben:

Tabelle 11.1 Windows Botschaftenklasse

Botschaftenklasse	Beschreibung
All Messages	Alle Botschaften, die mit WM_ beginnen.
Mouse	Botschaften, die durch ein Maus-Ereignis erzeugt werden (zum Beispiel *WM_LBUTTONDOWN* und *WM_MOUSEMOVE*).
Window	Botschaften vom Window Manager (z.B. *WM_PAINT* und *WM_CREATE*).
Input	Botschaften, die durch ein Tastaturereignis oder einen Zugriff des Benutzers auf ein Systemmenü, eine Bildlaufleiste oder ein Größenänderungsfeld erzeugt werden (z.B. *WM_KEYDOWN* und *WM_SYSCOMMAND*).
System	Botschaften, die durch eine systemweite Änderung erzeugt werden (z.B. *WM_FONTCHANGE* und *WM_SPOOLERSTATUS*).
Initialization	Botschaften, die erzeugt werden, wenn eine Anwendung ein Dialogfenster oder ein Fenster erzeugt (z.B. *WM_INITDIALOG* und *WM_INITMENU*).
Clipboard	Botschaften, die erzeugt werden, wenn eine Anwendung versucht, auf die Zwischenablage eines Fensters in einer anderen Anwendung zuzugreifen (z.B. *WM_DRAWCLIPBOARD* und *WM_SIZECLIPBOARD*).
DDE	(Dynamic Data Exchange) Diese Botschaften werden erzeugt, wenn Daten zwischen den Fenstern von Anwendungen ausgetauscht werden (z.B. WM_DDE_INITIATE und WM_DDE_ACK).
Non-client	Botschaften, die von Windows zur Verwaltung des Non-Client-Bereichs eines Anwendungsfensters erzeugt werden (z.B. *WM_NCHITTEST* und *WM_NCCREATE*).
Other	Alle Botschaften, die mit WM_ beginnen und in keiner der anderen Kategorien enthalten sind, wie z.B. Botschaften eines benutzerdefinierten Dialogelements.
Single Message	Eine einzelne, mit WM_ beginnende Botschaft, die Sie protokollieren oder bei der Sie unterbrechen möchten.

Zur Verfolgung einer einzelnen Botschaft wählen Sie *Single Message* und geben den Namen der Botschaft oder eine Nummer ein. Wenn Sie einen Namen eingeben, dürfen Sie nur Großbuchstaben verwenden.

Die Standardaktion ist das Protokollieren von Botschaften. Die andere mögliche Aktion, das Unterbrechen eines Programms, ist das Gleiche wie das Setzen eines Breakpoints für eine Botschaft.

Wenn Sie zum Beispiel die Botschaft *WM_PAINT* verfolgen möchten und das Programm jedesmal anhalten soll, wenn diese Botschaft an ein von Ihnen im Ausschnitt *Window Selection* ausgewähltes Fenster übertragen wird, gehen Sie wie folgt vor:

1. Selektieren Sie den Ausschnitt oben rechts, den Ausschnitt *Message Class* .

2. Rufen Sie das lokale Menü auf und wählen Sie *Add*.

3. Im Dialogfenster wählen Sie *Break* im Menüpunkt *Action* und die Option *Single message* im Menüpunkt *Message class*.

4. Geben Sie *WM_PAINT* im Eingabefeld für den Botschaftennamen ein und drücken Sie Return.

In Abbildung 11.1 sehen Sie, wie das Fenster *Windows Messages* aussieht, nachdem diese Einstellungen ausgewählt wurden und eine Botschaft empfangen wurde.

Löschen einer Botschaftenklasse Zum Löschen einer Botschaftenklasse plazieren Sie dem Cursor auf den gewünschten Eintrag. Dann wählen Sie entweder *Remove* aus dem lokalen Menü oder Sie drücken Entf, Strg R oder Strg Y.

Mit *Delete All* im lokalen Menü oder mit Strg D löschen Sie alle Klassen.

Wenn Sie alle Klassen gelöscht haben, ist die Standardklasse *Log all messages* eingestellt. Diese Klasse kann nicht mit *Remove* oder *Delete All* gelöscht werden.

Hinweise zu Windows-Botschaften

Wenn Sie Botschaften für mehrere Fenster anzeigen wollen, sollten Sie nicht alle Botschaften protokollieren. Stattdessen sollten Sie sich auf bestimmte Botschaften oder eine bestimmte Klasse von Botschaften für die einzelnen Fenster beschränken. Wenn Sie alle Botschaften protokollieren, kann es passieren, daß Ihr System aufgrund der großen Menge an Botschaften, die zwischen Windows und TDW ausgetauscht werden, abstürzt.

Beim Setzen eines Breakpoints für Mausnachrichten ist zu beachten, daß auf eine Botschaft der Art »Maus gedrückt« eine Botschaft der Art »Maus losgelassen« folgen muß, damit die Tastatur wieder angesprochen werden kann. Diese Einschränkung bedeutet, daß Sie bei der Rückkehr zu Ihrem Anwendungsprogramm die Maustaste unter Umständen mehrmals hintereinder drücken müssen, damit Windows eine Botschaft »Maus gedrückt« erhält. Ob Windows die Botschaft erhalten hat, können Sie dem linken unteren Ausschnitt des Fensters *Window Messages* entnehmen.

Falls Sie den Namen eines Handles als Prozedur angeben, wird diese Eingabe von TDW ohne Fehlermeldung akzeptiert. Wenn Sie jedoch Ihr Programm starten, protokolliert TDW keine Botschaften. Falls TDW keine Botschaften aufzeichnet, nachdem Sie ein Handle eingestellt haben, geben Sie das Handle erneut ein und achten bitte darauf, daß der Schalter *Handle* ausgewählt ist.

Anzeigen von Botschaften

Botschaften werden im unteren Ausschnitt des Fensters *Windows Messages* dargestellt. Dieser Ausschnitt kann bis zu 200 Botschaften aufnehmen.

Wenn Sie die Botschaften in einer Datei abspeichern wollen, müssen Sie eine Datei für das *Log*-Fenster aufmachen (nehmen Sie dazu den Befehl *View/Log File* und wählen Sie den Befehl *Open Log File* aus dem lokalen Menü). Gehen Sie dann in den Ausschnitt *Messages* zurück und setzen Sie *Send To Log Window* im lokalen Menü auf *Yes*.

Um alle Botschaften aus dem Ausschnitt zu löschen, wählen Sie *Erase Log* aus dem lokalen Menü. Die Botschaften, die in das *Log*-Fenster geschrieben wurden, werden von diesem Befehl nicht berührt.

Speicher- und Modullisten

Um den Inhalt des globalen oder lokalen Heap oder die Module für Ihr Windows-Programm aufzulisten, öffnen Sie zuerst mit dem Befehl *View/Log* das Fenster *Log* und aktivieren das lokale Menü. Der letzte Befehl im lokalen Menü des *Log*-Fensters ist *Display Windows info* . Durch Anwahl dieses Befehls öffnen Sie das Dialogfenster *Windows Information* , wo Sie die Art und den Beginn der Liste festlegen können.

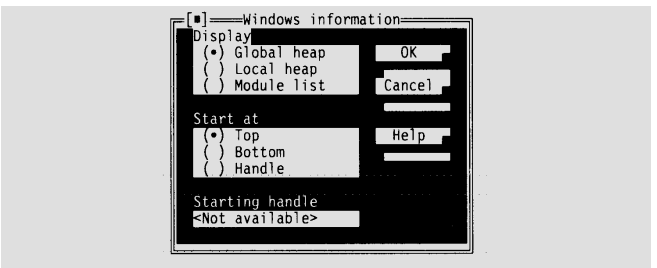


Abbildung 11.6 Das Dialogfenster Windows Information

Wenn Sie die Option *Global Heap* wählen, können Sie sich die Liste von oben nach unten, unten nach oben oder aber ab einem Punkt, der durch ein Starthandle definiert wird, anzeigen lassen.

Ein Starthandle ist der Name eines globalen Speicherhandle, das Ihre Anwendung durch den Aufruf einer Windows- Speicherzuweisungsroutine, wie z.B. *GlobalAlloc* , eingerichtet hat. Die Auswahl eines Starthandle veranlaßt TDW, das an der Adresse des Handle liegende Objekt sowie die nächsten vier Objekte, die im Heap folgen, anzuzeigen.

Auflisten des globalen Heap

Der globale Heap ist der globale Speicherbereich, den Windows für alle Anwendungen zur Verfügung stellt. Wenn Sie Ressourcen, wie Symbole, Bitmaps, Dialogfenster und Fonts zuweisen oder wenn Sie Speicherplatz mit der GlobalAlloc Funktion allozieren, benutzt Ihre Anwendung den globalen Heap.

Zur Anzeige der Datenobjekte im globalen Heap wählen Sie *Global Heap* im Fenster *Windows-Information* und klicken dann den Schalter *OK*. Die Datenobjekte werden im *Log*-Fenster aufgelistet. Weil dabei wahrscheinlich die Höchstzahl der Zeilen, die im *Log*-Fenster beschrieben werden können (Standard ist 50 Zeilen), überschritten wird, sollten Sie den Inhalt entweder in eine Log-Datei schreiben (mit Hilfe des lokalen Menüs des *Log*-Fensters) oder die Anzahl der beschreibbaren Zeilen im *Log*-Fenster erhöhen (benützen Sie TDWINST). Die höchste Anzahl von Zeilen, die Sie einstellen können, ist 200.

Die folgende Tabelle zeigt als Beispiel zwei Zeilen einer Ausgabe des globalen Heap und erklärt die einzelnen Felder dieser Ausgabe:

Tabelle 11.2 Format einer Liste des globalen Heap

Beispiel für die Ausgabe des globalen Heap	
0EC5	00000040b PDB (0F1D)
053E(053D)	00002DC0b GDI DATA MOVEABLE LOCKED=00001 PGLOCKED=0001

Feld	Beschreibung
0EC5 053E	Entweder ein Handle für das Speicherobjekt, als 4-stelliger Hexwert, oder das Wort FREE, wodurch ein freier Speicherblock angezeigt wird.
(053D)	Ein Speicherselektor, der auf einen Eintrag in der globalen Deskriptortabelle zeigt. Der Selektor wird nicht angezeigt, wenn er den gleichen Wert wie das Speicherhandle hat.
00000040b 00002DC0b	Eine hexadezimale Zahl, die die Länge des Segments in Byte angibt.
PDB GDI	Der Code, der ein Segment reserviert, normalerweise eine Anwendung oder ein Bibliotheksmodul. Ein PDB ist ein Prozeßdeskriptorblock, auch bekannt als Programmsegmentpräfix (PSP).
(0F1D)	Ein Handle, das den Eigner eines PDB angibt.
DATA	Der Typ des Speicherobjekts. Es gibt folgende Typen:
DATA	Anwendung oder DLL
CODE	Codesegment einer Anwendung oder DLL
PRIV	Entweder ein Systemobjekt oder globale Datenfür eine Anwendung oder DLL
MOVEABLE	Ein Speicherverwaltungsattribut. Ein Objekt kann die Attribute FIXED, MOVEABLE oder MOVEABLE DISCARDABLE haben.
LOCKED=00001	Anzahl der Sperren, die an einem MOVEABLE oder MOVEABLE-DISCARDABLE Objekt mit der Funktion GlobalLock oder LockData gesetzt wurden.
PGLOCKED=0001	Im erweiterten 386er-Modus die Anzahl von Pagelocks, die mit der GlobalPageLock-Funktion an einem Objekt gesetzt wurden. Bei einem gesetzten Pagelock kann Windows keine der 4 Kilobyte-Seiten eines Objekts auf die Platte auslagern.

Auflisten des lokalen Heap

Der lokale Heap ist ein privater Speicherbereich der Anwendung. Andere Windows-Anwendungen, einschließlich anderer Instanzen derselben Anwendung, können nicht auf den lokalen Heap zugreifen.

Ein Programm hat nicht unbedingt einen lokalen Heap. Windows legt einen

lokalen Heap an, wenn in der Definitionsdatei der Anwendung die Anweisung HEAPSIZE steht oder die Anwendung die Funktion *LocalAlloc* verwendet.

Zur Anzeige der Datenobjekte im lokalen Heap wählen Sie *Local Heap* im Fenster *Windows Information* und klicken dann den Schalter OK. Die Datenobjekte werden im *Log*-Fenster aufgelistet.

Die Liste kann leicht die Standardgröße des *Log*-Fensters übersteigen. Für die Verwendung von *Log*-Dateien und die Erhöhung der Zeilenzahl des *Log*-Fensters gilt das gleiche wie für den globalen Heap.

Die folgende Tabelle zeigt eine typische Zeile der Ausgabe des lokalen Heap und deren Format:

Tabelle 11.3 Format einer Liste des lokalen Heap

Ausgabe des lokalen Heap		
05CD: 0024 BUSY (10AF)		
Feld	**Beschreibung**	
05CD:	Der Offset des Objekts im lokalen Datensegment.	
0024	Die Länge des Objekts in Byte.	
BUSY	Der Zustand des Speicherobjekts, und zwar:	
	FREE	Ein nicht zugewiesener Speicherblock.
	BUSY	Ein zugewiesenes Objekt.
(10AF)	Ein lokales Speicherhandle für das Objekt.	

Modulliste

Zur Anzeige der Liste von Tasks und DLL-Modulen, die von Windows geladen wurden, wählen Sie im Dialogfenster *Windows Information* die Option *Module List* an und klicken dann den Schalter *OK*. Die Module werden im *Log*-Fenster aufgelistet.

Die Liste kann leicht die Standardgröße des *Log*-Fensters übersteigen. Für die Verwendung von *Log*-Dateien und die Erhöhung der Zeilenzahl des *Log*-Fensters gilt das gleiche wie für den globalen Heap.

Die folgende Tabelle zeigt drei Beispielzeilen einer Modulliste. Danach wird die letzte Zeile der Liste erklärt:

Tabelle 11.4 Format einer Windows-Modulliste

Beispiel für die Ausgabe einer Modulliste	
0985 TASK TDW	C:\TD\TDW.EXE
0E2D DLL WINDEBUG	C:\WIN3\WINDEBUG.DLL
0EFD TASK GENERIC	C:\TD\GENERIC.EXE
0EFD	Ein Handle für das Speichersegment, als 4-stelliger Hexwert.
TASK	Der Modultyp. Ein Modul kann eine TASK oder eine DLL sein.
GENERIC	Der Modulname.

Feld	Beschreibung
C:\TD\GENERIC.EXE	Der Pfad zu der ausführbaren Datei des Moduls

Das Debuggen von dynamischen Link-Bibliotheken (DLLs)

Eine DLL ist eine Windows-Bibliothek mit Routinen und Ressourcen. Diese Bibliothek wird nicht während der Programmübersetzung zu Ihrer Anwendung gelinkt, sondern zur Laufzeit. Das Linken zur Laufzeit erlaubt mehreren Anwendungen, sich eine einzige Kopie von Routinen, Daten oder Gerätetreibern zu teilen und damit Speicherplatz zu sparen. Wenn eine Anwendung, die eine DLL benützt, gestartet wird und die DLL ist noch nicht im Speicher, lädt Windows diese DLL, so daß die Anwendung auf die Einsprungspunkte der DLL zugreifen kann.

TDW kann eine DLL ohne Symboltabelle laden, allerdings nur in ein CPU-Fenster und nur, wenn Sie in einem CPU-Fenster debuggen. Wenn Sie eine Anwendung mit eingebunden DLLs in TDW laden, stellt TDW fest, ob überhaupt und welche dieser DLLs Symboltabellen besitzen (d.h. mit eingeschalteter Debug-Option kompiliert wurden), und verfolgt die DLLs. Wenn TDW zur Laufzeit Ihrer Anwendung einen Zugriff auf einen Einsprungspunkt einer dieser DLLs feststellt, lädt TDW die zugehörige Symboltabelle und den Quelltext und positioniert die Modul-Zeilenmarkierung auf den Anfang der von Ihrer Anwendung aufgerufenen DLL-Routine. Die DLL ist dann im Fenster *Module* verfügbar, genau wie Ihr Anwendungs-Quelltext.

Wenn die DLL-Routine beendet ist, lädt TDW wieder die Symboltabelle und den Quelltext Ihrer Anwendung, markiert die Zeile, in der die nächste Anweisung nach dem Zugriff auf den DLL-Einsprungspunkt steht.

Falls Sie ein Programm mit Hilfe von F7 oder F8 schrittweise untersuchen, kann es sein, daß TDW nicht zur aufrufenden Routine in Ihrem Programm zurückkehrt, da die DLL eventuell eine Windows-Funktion aufruft. In diesem Fall wird Ihr Programm so ausgeführt, als ob Sie F9 gedrückt hätten. Dieses Verhalten finden Sie sehr häufig im Startup-Code von DLLs. Um die Rückkehr zu Ihrem Programm sicherzustellen, setzen Sie einen Breakpoint hinter den Aufruf der DLL, bevor Sie den DLL-Aufruf debuggen. Bei der Untersuchung des DLL-Startup-Codes setzen Sie den Breakpoint auf die erste Zeile in Ihrem Anwendungsprogramm.

Weil ein großer Teil der DLL-Debug-Prozedur automatisch abläuft, müssen Sie nicht spezifizieren, welche DLLs geladen werden sollen. Möglicherweise möchten Sie jedoch andere Aufgaben ausführen, wie zum Beispiel:

- Hinzufügen einer DLL in der DLL-Liste.
- Setzen von Breakpoints, Watches etc. in einer DLL.
- Angeben, welche DLLs von TDW nicht geladen werden sollen.
- Den DLL Startup-Code debuggen.

Zur Ausführung einer dieser Aufgaben öffnen Sie mit dem Befehl *View/ Modules* das Dialogfenster *Load Modules or DLLs*. (Dieses Fenster kann auch durch Drücken von F3 aufgerufen werden.)

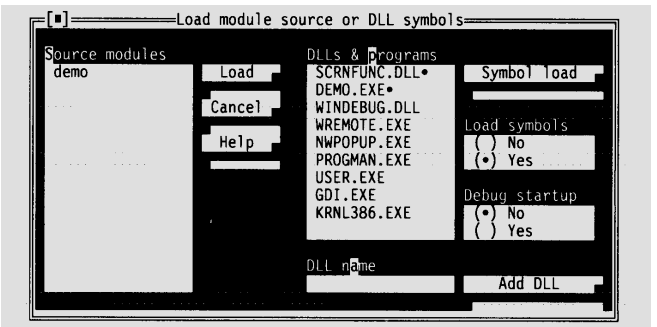


Abbildung 11.7 Das Dialogfenster Load Modules or DLLs

Benutzung des Dialogfenster *Load Modules or DLLs* Mit diesem Dialogfenster können Sie zwei Dinge tun:

- In ein anderes Quellmodul Ihrer Anwendung wechseln;
- Vorgänge mit DLLs und .EXE-Dateien ausführen (z.B. Laden einer Symboldatei und Quelldatei).

Wechseln der Quellmodule Wenn Sie eine Anwendung debuggen, die aus mehreren in eine .EXE-Datei eingebundenen Quellmodulen besteht, und Sie auf ein anderes, gerade nicht im *Module*-Fenster befindliches Modul dieser Anwendung zugreifen möchten, können Sie das Dialogfenster *Load Modules or DLLs* öffnen und ein Modul aus der linken Liste, der Quellmodulliste, auswählen. Alle in Ihre Anwendung eingebundenen Module stehen in dieser Liste.

Um ein Modul auszuwählen, müssen Sie es zuerst markieren. Dann drücken Sie entweder die Eingabetaste CR, klicken zweimal mit der Maus oder wählen das Feld *Load* an. TDW zeigt das Fenster *Module* mit dem neuen Quellmodul an.

Arbeiten mit DLLs und Programmen Wenn Sie eine Anwendung debuggen, mit der eine oder mehrere DLLs verknüpft sind (das ist bei jeder Windows-Anwendung der Fall), und Sie das Dialogfenster *Load Modules or DLLs* öffnen, werden in der DLL- und Programmliste (die rechte Liste) DLLs und .EXE-Dateien angezeigt. Diese Liste umfaßt alle DLLs und .EXE-Dateien, die Windows momentan geladen hat, sowie alle DLLs, die gestartet werden, wenn Ihre Anwendung startet. Nicht eingeschlossen sind DLLs, die Ihre Anwendung mit einem *LoadLibrary* -Aufruf starten, es sei denn, eine dieser DLLs wurde schon von Ihrem Programm oder von Windows geladen.

Die Einträge am Anfang der Liste, die rechts mit einem Oval markiert sind, sind die .EXE-Datei Ihrer Anwendung und die DLLs mit Symboltabellen, die von Ihrer Anwendung aufgerufen werden. Wenn Sie nichts anderes einstellen, lädt TDW für jede markierte DLL, die von Ihrer Anwendung aufgerufen wird, die Symboltabelle und den Quelltext. Darüber hinaus lädt TDW automatisch die

Symboltabelle und den Quelltext jeder DLL, die von Ihrer Anwendung über einen *LoadLibrary* -Aufruf gestartet wird, obwohl dieses DLL zuerst nicht in der Liste steht. (Nach dem Laden durch TDW erscheint auch diese DLL in der Liste).

Mit den Schaltern rechts von der Liste können Sie Einstellungen für die markierte DLL oder Anwendung vornehmen. In dem Textfeld unter der Liste können Sie eine weitere DLL hinzufügen. Folgendes können Sie einstellen:

Tabelle 11.5 Die Schalter im Dialogfenster DLLs & Programs List

Schalter	Beschreibung
Symbol load	Laden der Symboltabelle und Quelldateien für die DLL oder Anwendung. Dieser Befehl wird unabhängig von der Einstellung unter *Laod Symbols* ausgeführt. Dadurch verändert sich der Inhalt des Fensters *Module*, so daß Sie Breakpoints, Windows-Botschaften etc. für die DLL oder Anwendung einstellen können.
Load symbols (*No*/*Yes*)	Legen Sie fest, ob Symboltabelle und Quelltext der DLL geladen werden, wenn die Anwendung die DLL aufruft. Mit dieser Option können Sie verhindern, daß TDW die Symboltabelle und den Quelltext einer DLL, die Sie nicht debuggen wollen, lädt. Die Standardeinstellung ist *Yes*. Die Auswahl von *Yes* erkennen Sie durch ein Oval neben dem Namen der DLL. Wenn Sie ein Programm neu laden, wird *Load Symbols* bei allen DLLs und Modulen auf *Yes* gesetzt, die zuvor auf *No* eingestellt waren.
Debug startup (*No*/*Yes*)	Stellen Sie ein, ob Sie den Startup-Code der DLL debuggen möchten. Die Standardeinstellung ist *No*. Bei der Auswahl von *Yes* wird das Modul oder die DLL durch zwei Ausrufezeichen (!!) gekennzeichnet. Diese Schalter werden nur für DLLs ausgewertet. Um den Startup-Code des Hauptprogramms zu untersuchen müssen Sie TDW mit der Kommandozeilenoption *-i* starten.
DLL Name	Geben Sie den Namen einer DLL ein, die sich nicht in der Liste *DLLs & Programs* befindet, damit Sie diese der Liste hinzufügen können. Sie können jede beliebige Dateierweiterung verwenden. Wenn Sie eine DLL hinzugefügt haben, können Sie einen der drei vorhergehenden Befehle darauf anwenden. Wenn nötig, können Sie den ganzen Pfadnamen angeben.
Add DLL	Nimmt die DLL aus dem Textfeld in die Liste *DLLs & Programs* auf. Bei jeder DLL, die manuell eingefügt wird, werden *Load Symbols* und *Debug Startup* auf *Yes* eingestellt.

Hinzufügen einer DLL zur Liste *DLLs & Programs*

Bevor Sie Debug-Optionen einstellen können, den DLL-Startup-Code debuggen oder einstellen können, daß die Symboltabelle und der Quelltext einer DLL nicht geladen werden, muß die entsprechende DLL zuerst in der Liste *DLLs & Programs* stehen. Eine DLL, auf die Ihre Anwendung zugreift, muß nach dem Laden Ihrer Anwendung nicht unbedingt in der Liste stehen, weil TDW nur

solche DLLs kennt, die in den Startup-Code Ihrer Anwendung eingebunden sind. Ihre Anwendung kann eine DLL auch mit dem Befehl *LoadLibrary* starten; TDW erkennt diese DLL erst dann, wenn Ihre Anwendung *LoadLibrary* aufruft.

In diesem Abschnitt ist von zwei verschiedenen Startup-Codes die Rede: der Startup-Code Ihrer Anwendung und der DLL-Startup-Code. Manche DLLs werden im Startup-Code Ihrer Anwendung gestartet. Wenn Ihre Anwendung eine DLL aufruft, dann wird der Startup-Code der DLL ausgeführt.

Es gibt auch zwei Arten von DLL-Startup-Codes. Diese werden im Abschnitt »Das Debuggen von DLL-Startup-Codes« erklärt.

Wenn Sie eine DLL in die Liste *DLLs & Programs* aufnehmen möchten, öffnen Sie das Dialogfenster *Load Modules or DLLs* (drücken Sie F3 oder verwenden Sie den Befehl *View/Modules*). In das Textfeld schreiben Sie den Namen der DLL (mit dem vollständigen Pfadnamen, wenn nötig) und betätigen dann den Schalter *Add DLL*.

Einstellen von Debug-Optionen in einer DLL

Zum Setzen von Breakpoints, Watches oder anderen Debug-Optionen für eine DLL, öffnen Sie das Dialogfenster *Load Modules or DLLs* (drücken Sie F3 oder verwenden Sie den Befehl *View/Modules*). Suchen Sie in der Liste *DLLs & Programs* nach der gewünschten DLL und markieren Sie sie. Der Befehl *Symbol Load* öffnet das Fenster *Module* mit der DLL. Im Fenster *Module* können Sie die gewünschten Einstellungen an der DLL vornehmen.

Steuern, welche DLL-Symboltabellen von TDW geladen werden

Standardmäßig lädt TDW die Symboltabelle und den Quelltext jeder DLL, auf die Ihre Anwendung zugreift, wenn diese DLL eine kompatible Symboltabelle besitzt. Eine DLL-Symboltabelle ist kompatibel mit TDW, wenn die DLL mit Turbo C++ für Windows und eingeschalteten Debug-Informationen compiliert wurde.

Weil es Zeit kostet, die Symboltabelle einer DLL zu laden und dann, wenn die DLL-Routine beendet ist, die Symboltabelle der ursprünglichen Anwendung zu laden, ist es sinnvoll, für DLLs, die Sie nicht debuggen wollen, die TDW-Standardfunktion auszuschalten. Dazu öffnen Sie das Dialogfenster *Load Modules or DLLs* (drücken Sie F3 oder verwenden Sie den Befehl *View/Modules*), suchen in der Liste *DLLs & Programs* nach der gewünschten DLL, markieren diese und setzen dann den Schalter *Load Symbols* auf *No*.

Das Debuggen von DLL-Startup-Code

Standardmäßig untersucht TDW keine DLL-Startup-Codes und lädt die Symboltabelle einer DLL nur, wenn Ihre Anwendung auf einen DLL-Einsprungspunkt zugreift. TDW öffnet dann eines der Fenster *Module* oder *CPU*. Die Zeilenmarkierung steht auf dem Anfang der von der Anwendung aufgerufenen DLL-Routine.

Wenn Sie wollen, können Sie mit TDW auch den DLL-Startup-Code debuggen. TDW kann zwei Arten von DLL-Startup-Codes debuggen:

- Den Initialisierungscode, der unmittelbar auf *LibMain* folgt (Standardvorgabe).

- Den Assemblercode, der in die DLL eingebunden ist, die den Startup durchführt und emulierte Mathematikpackages für das Modell, unter dem die DLL läuft, enthält. (Wird beim Start von TDW durch die Kommandozeilenoption *-l* ausgewählt).

Nachdem Sie für eine oder mehrere DLLs ein Debuggen des Startup-Codes festgelegt haben, lädt TDW für jede DLL die Symboltabelle, wenn der Startup-Code Ihrer Anwendung die DLL startet oder wenn Ihre Anwendung LOADLIBRARY aufruft.

Wenn Sie Startup-Debuggen nach dem Laden Ihrer Anwendung einstellen, kann es passieren, daß TDW sich nicht wie erwartet verhält. Das liegt daran, daß einige oder alle der DLLs schon geladen worden sind. Deshalb sollten Sie das Debuggen von Startup-Codes einstellen indem Sie:

- die DLLs vor dem Laden Ihrer Anwendung festlegen.

- Ihre Anwendung laden, die DLLs für das Startup-Debuggen bestimmen und anschließend einen Neustart Ihrer Anwendung durchführen (Strg F2 oder *Run/Program Reset*).

Ausführung des Debuggens von Startup-Codes Von diesen Voraussetzungen ausgehend, verfahren Sie wie folgt, um das Startup-Debuggen für eine oder mehrere DLLs anzugeben und den Startup-Code dieser DLLs zu debuggen:

1. Öffnen Sie das Dialogfenster *Load Modules or DLLs* (drücken Sie F3 oder verwenden Sie den Befehl *View/Modules*).

2. Ist kein Programm geladen, machen Sie weiter mit Schritt 5. Ansonsten wählen Sie eine DLL in der Liste *DLLs & Programs* aus und markieren diese.

3. Setzen Sie den Schalter *Debug Startup* auf *Yes*.

4. Wiederholen Sie die Schritte 2 und 3 so oft, bis Sie für alle gewünschten DLLs Startup-Debuggen eingestellt haben.

5. Wenn eine von Ihnen gewünschte DLL nicht in der Liste steht oder überhaupt keine DLLs enthalten sind (weil Sie Ihre Anwendung noch nicht geladen haben), geben Sie die Namen der gewünschten DLLs einzeln in das Textfeld ein und betätigen Sie jeweils den Schalter *Add DLL*, um die DLL in die Liste aufzunehmen.

6. Wenn Sie alle DLLs, deren Startup-Code Sie debuggen wollen, festgelegt haben, ist der nächste Schritt entweder das Laden Ihrer Anwendung mit dem Befehl *File/Load* (wenn Sie sie noch nicht geladen haben) oder das Wiedereinladen Ihrer Anwendung mit dem Befehl *Run/Program Reset* (wenn Sie sie bereits vor dem Einstellen des Startup-Debuggens geladen hatten).

7. Bevor Sie das Programm starten, sollten Sie durch das Setzen von Breakpoints sichergestellen, daß die DLLs zu Ihrem Anwendungsprogramm zurückkehren, nachdem die Initialisierung abgeschlossen ist. Mit dem Quelltext Ihres Programm im *Module*-Fenster gehen Sie wie folgt vor:

- Setzen Sie einen Breakpoint auf die erste Zeile Ihres Programms.

■ Wenn Sie den Initialisierungscode einer der DLLs untersuchen wollen, die mit einem Aufruf von *LoadLibrary* geladen werden, setzen Sie einen Breakpoint direkt hinter die Zeile mit diesem Aufruf.

8. Da jede DLL von Ihrer Anwendung gestartet wird, öffnet TDW entweder das Fenster *Module* mit *LibMain* der DLL (Standard) oder das Fenster *CPU* am Anfang des Assemblercode-Listings für die Startup-Bibliothek (weil Sie TDW mit der Option *-l* aufgerufen haben).

9. Nachdem das Debuggen des Startup-Codes einer DLL beendet ist, durchlaufen Sie durch Drücken von F9 das Ende des Startup-Codes und kehren zur Anwendung zurück. Wenn Sie weitere DLLs für das Debuggen des Startup-Codes vorgesehen haben, zeigt TDW deren Startup-Code an, wenn sie von Ihrer Anwendung gestartet werden.

Achtung: Stellen Sie sicher, daß Sie bis zum Ende des DLL Startup-Codes gehen, bevor Sie die aktuelle Anwendung wieder oder eine neue Anwendung laden. Wenn Sie das nicht tun, ist es möglich, daß der nur teilweise ausgeführte DLL-Startup-Code Windows aufhängt und Sie neu booten müssen.

Die Umwandlung von Speicherhandles in Adressen

Windows benützt Speicherhandles anstatt Adressen für Speicherobjekte, weil es seine eigene Speicherverwaltung hat und die physikalische Lage von Objekten im Speicher verändern kann. Wenn Sie die tatsächliche Adresse benötigen, auf die sich ein Speicherhandle bezieht, können Sie die in TDW eingebauten Typumwandlungssymbole `lh2fp` (für ein lokales Handle) und `gh2fp` (für ein globales Handle) für die Dereferenzierung eines Handles benützen.

Die Typumwandlungssymbole in TDW werden genauso verwendet wie die normalen Typumwandlungssymbole für Zeiger in C. Das lokale Speicherhandle `hLocalMemory` könnte zum Beispiel auf zwei Arten umgewandelt werden:

■ Sie könnten das Fenster Data/Inspect benützen, um den folgenden Ausdruck auszuwerten:

```
(lh2fp) hLocalMemory
```

■ Sie könnten den Befehl *Type Cast* im lokalen Menü des Inspector verwenden und `lh2fp` als Typ eingeben.

In beiden Fällen liefert der Ausdruck das erste Zeichen des Speicherblocks, auf den `hLocalMemory` zeigt.

Sie können auch beide Techniken für komplexe Umwandlungen kombinieren - zum Beispiel, eine 2-Stufen-Umwandlung von einem Handle in einen FAR-Zeichen-Zeiger und dann in einen FAR-Zeiger auf die Daten im Speicher:

```
(Mystruct far *)(Lh2fp)hLocalMemory
```

12. Debuggen auf Assembler-Ebene

Dieses Kapitel richtet sich an Programmierer, die mit der Assembler-Programmierung der 80x86-Prozessoren vertraut sind. Sie müssen die in diesem Kapitel beschriebenen Möglichkeiten nicht verwenden, wenn Sie Ihre Programme debuggen wollen, obwohl einige raffinierte Fehler mit den hier beschriebenen Methoden leichter gefunden werden können.

Wir erklären, wann es für Sie sinnvoll ist, beim Debuggen auf die Assembler-Ebene zurückzugreifen. Anschließend beschreiben wir das *CPU*-Fenster mit dem integrierten Disassembler und Assembler. Sie werden sehen, wie man reine Hex-Daten untersucht und verändert, wie der Stack mit den Funktionsaufrufen eingesetzt wird und wie CPU-Register und CPU-Flags untersucht und verändert werden.

Grenzen des Debuggens auf Quellcode-Ebene

Wenn Sie ein Programm debuggen, dann greifen Sie auf den Code und die Daten Ihres Programms meistens auf der Quellcode-Ebene zu. Sie verwenden die Symbolbezeichnungen genau so, wie Sie sie in Ihrem Quellcode angegeben haben. Sie wandern durch Ihr Programm, indem Sie Teile des Quellcodes abarbeiten.

Manchmal aber kann man ein Problem nur durch genaue Betrachtung der vom Compiler erzeugten Befehle, des Inhalts der CPU-Register und des Stack-Inhalts erfassen, wie zum Beispiel:

- Anschauen des Inhalts eines Speicherbereichs, der von einem Protected-Mode Selektor referenziert wird.

- Anschauen der Befehle, die der Compiler aus einer Quellcodezeile erzeugt hat, sowie Anschauen des Inhalts des Stack und der CPU-Register.

- Abarbeiten des Windows-Codes um herauszufinden, an welcher Stelle Ihr Programm angehalten hat.

Um eine dieser Funktionen auszuführen, müssen Sie das CPU-Fenster verwenden. Zudem erhalten Sie Einblick über die Speicherverwaltung unter Windows. Dazu müssen Sie aber mit den 80x86-Prozessoren vertraut sein und eine

Vorstellung darüber haben, wie der Compiler Ihren Quellcode in Maschinensprachebefehle umwandelt. Da es sehr viele ausgezeichnete Bücher über die interne Funktionsweise der CPU gibt, werden wir hier nicht ins Detail gehen. Sie können sehr schnell lernen, wie der Compiler Ihren Quellcode in die Maschinensprachebefehle umwandelt, indem Sie sich die Befehle, die für jede einzelne Zeile im Quellcode erzeugt werden, genau anschauen.

In C und Pascal zum Beispiel können im Quellcode Zeilen geschrieben werden, die sehr viele Aktionen auf einmal ausführen. Da der Debugger jeweils eine ganze Zeile und nicht einen einzelnen Ausdruck abarbeitet, wenn man schrittweise durchs Programm geht, kann in der Auswertung nicht, wie vielleicht gewünscht, jeder einzelne Schritt nachvollzogen werden. Indem jeder Maschinensprachebefehl einzeln abgearbeitet wird, kann man alle Zwischenresultate kontrollieren.

Das CPU-Fenster

Das *CPU*-Fenster gibt einen vollständigen Überblick über den Zustand der CPU. Sie können die Bits und Bytes, aus denen die Daten und der Code Ihres Programms bestehen, direkt untersuchen und verändern. Mit dem integrierten Assembler kann das Programm im Code-Ausschnitt temporär gepatcht werden. Die Befehle werden wie in einem Assemblerquelltext eingegeben. Sie können außerdem auf alle Bytes der damit zusammenhängenden Datenstrukturen zugreifen und diese in einer Reihe von Formaten darstellen und verändern.

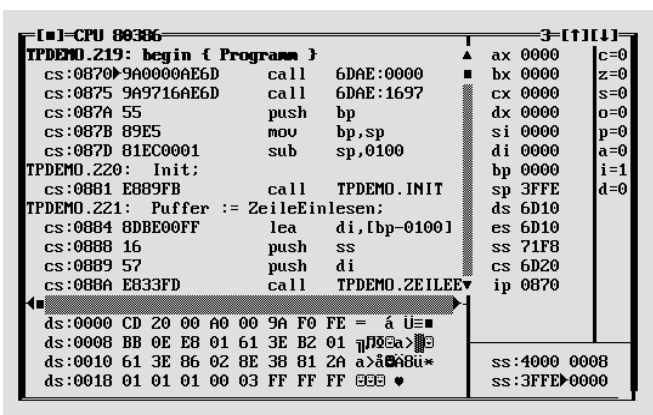


Abbildung 11.1
Das CPU-Fenster

Ein CPU-Fenster kann von der Menüleiste aus mit dem Befehl View/CPU geöffnet werden. Je nachdem, was Sie im aktuellen Fenster dargestellt haben, erscheint das CPU-Fenster mit der entsprechenden Stelle im Code-, Daten- oder Stack-Bereich. Dies ist eine bequeme Möglichkeit, an der jeweiligen Stelle im Code-, Daten- oder Stack-Bereich unter die Oberfläche zu gehen und alle

Informationen detailliert anzuschauen. Der folgenden Tabelle können Sie entnehmen, wo der Cursor stehen wird, wenn Sie den CPU-Befehl eingeben:

Aktuelles Fenster	CPU-Ausschnitt	Position
Stack-Fenster	Stack	Aktueller SS:SP
Module-Fenster	Code	Aktueller CS:IP
Variables-Fenster	Daten*	Adresse des Datenobjekts
Watches-Fenster	Daten*	Adresse des Datenobjekts
Inspector-Fenster	Daten*	Adresse des Datenobjekts
Breakpoint (kein globaler)	Code	Adresse des Breakpoints

* Code-Ausschnittwenn das Datenobjekt eine Routine ist.

Mit TDW gelangen Sie automatisch in das CPU-Fensters, wenn TDW wieder die Kontrolle über das Programm erhält und der ausgeführte Windows-Code oder die DLL keine Debug-Informationen enthält.

Das *CPU*-Fenster besteht aus sechs Ausschnitten. Um von einem Ausschnitt zum nächsten zu gehen, drücken Sie Tab oder Umschalt Tab . Mit der Maus brauchen Sie nur in den gewünschten Ausschnitt zu klicken. Die oberste Zeile des *CPU*-Fensters zeigt den Prozessor-Typ Ihres System an (8086, 80286, 80386 oder i486).

Der linke obere Ausschnitt (Code-Ausschnitt) zeigt den disassemblierten Programmcode, in dem zwischendurch die entsprechenden Zeilen des Quellcodes stehen. Der zweite Ausschnitt oben (Registerausschnitt) zeigt den Inhalt der CPU-Register. Der rechte Ausschnitt ist der Statusausschnitt, in dem der Zustand der acht CPU-Flags dargestellt wird. In dem Ausschnitt auf der linken Seite zwischen Code- und Datenausschnitt (dem *Selector*-Ausschnitt), werden alle Windows-Selektoren aufgeführt und deren Inhalt angegeben. Der linke untere Ausschnitt (Datenausschnitt) enthält den reinen Hex-Dump eines ausgewählten Speicherbereichs. Der rechte untere Ausschnitt (Stack-Ausschnitt) zeigt den Inhalt des Stacks.

Wie alle anderen lokalen Menüs wird auch das lokale Menü des Code-Ausschnitts mit der Tastenkombination Alt F10 aktiviert. Wenn die Tastenkürzel mit den Strg -Tasten eingeschaltet sind, kann mit Hilfe der Taste Strg und dem ersten Buchstaben des Befehls direkt der gewünschte Befehl aufgerufen werden.

In den Code-, Daten- und Stack-Ausschnitten läßt sich die erste dargestellte Adresse des Ausschnitts mit den Tastenkombinationen Strg ← und Strg → um ein Byte nach oben oder unten verschieben. Für eine geringfügige Anpassung der Darstellung ist das leichter als die Verwendung des Befehls *Goto*.

Der Code-Ausschnitt

In diesem Ausschnitt werden die disassemblierten Befehle an der von Ihnen vorgegebenen Adresse dargestellt. Es gibt folgende Möglichkeiten, eine Adresse auszuwählen:

■ Aktivieren Sie mit Alt F10 das lokale Menü des Code-Ausschnitts und verwenden einen dieser Befehle: *Goto*, *Origin*, *Follow*, *Caller* oder *Previous*.

■ Positionieren Sie den Cursor im *Selector*-Ausschnitt auf einen Code-Selektor und wählen *Examine* aus dem lokalen Menü, um sich den Inhalt des Selektors im Code-Ausschnitt anzeigen zu lassen.

Der linke Teil jeder disassemblierten Zeile enthält die Adresse des Befehls. Die Adresse wird entweder hexadezimal als Segment und Offset dargestellt, oder der Wert des Segments wird mit CS angegeben, wenn das Segment mit dem aktuellen Code-Segment übereinstimmt. Wenn das Fenster breit genug ist (gezoomt oder vergrößert), werden auch noch die Bytes der einzelnen Maschinensprachebefehle angegeben. Der disassemblierte Befehl erscheint auf der rechten Seite.

Wenn sich der hervorgehobene Befehl im Code-Ausschnitt auf eine Speicherstelle bezieht, wird die Adresse und der aktuelle Inhalt der Speicherstelle in der obersten Zeile des CPU-Fensters angezeigt. Daraus kann man ersehen, auf welche Speicherstelle sich ein Befehlsoperand bezieht und welcher Wert in dieser Speicherstelle gelesen oder überschrieben wird.

Der Disassembler

Im Codeausschnitt werden Ihre Programmbefehle automatisch disassembliert und dargestellt. Wenn eine Adresse einem globalen oder statischen Symbol beziehungsweise einer Zeilennummer entspricht, enthält die Zeile vor der disassemblierten Anweisung die Quellcodezeilen, wenn als Darstellungsart beim Befehl *Mixed* die Option *Yes* angegeben wurde. Wenn die Quellcodezeile der Symboladresse entspricht, wird diese auch hinter dem Symbol dargestellt.

Globale Symbole werden einfach in Form der Symbolbezeichnung angegeben. Die statischen Symbole erscheinen als Modulnamen, gefolgt von einem Nummernzeichen (#) oder einem Punkt (.). Zeilennummern werden hinter dem entsprechenden Modulnamen angegeben. Dazwischen steht noch ein Nummernzeichen (#) oder ein Punkt (.). Die Zeilennummern werden dezimal angegeben.

Wenn ein konstanter Operand dargestellt wird, kann man den Datentyp aus der Anzahl der Ziffern ermitteln: ein Byte hat zwei Ziffern, ein Wort besteht aus vier Ziffern.

Turbo Debugger kann 8087/80287/80387/80486-Koprozessoren automatisch entdecken und disassembliert diese Befehle entweder für den Koprozessor-Chip oder für den Emulator.

Der Befehl RETF deutet an, daß ein Rücksprung als FAR erfolgt. Der normale RET-Befehl bedeutet einen NEAR-Rücksprung.

Falls möglich, werden die Zieladressen von JMP- und CALL-Befehlen symbolisch dargestellt. Wenn CS:IP bei einem JMP- oder bedingten Sprungbefehl steht, deutet ein Pfeil nach oben und unten an, in welche Richtung die Verzweigung im Befehl durchgeführt wird. Speicheradressen, die in MOV, ADD und anderen Befehlen verwendet werden, werden ebenfalls symbolisch dargestellt.

Das lokale Menü des Code-Ausschnitts

Wenn Sie mit dem Cursor nicht im Codeausschnitt landen, können Sie die Tasten Tab und Umschalt Tab verwenden, um dorthin zu gelangen. Mit Alt F10 aktivieren Sie das lokale Menü.

Goto

Nachdem Sie den Befehl *Goto* (engl. gehe zu) ausgewählt haben, werden Sie nach der neuen Adresse gefragt. Sie können auch Adressen außerhalb Ihres Programmbereichs verwenden, um den Code im ROM-BIOS, in MS-DOS oder in speicherresidenten Programmen zu untersuchen. In Kapitel 9, »Ausdrücke«, erhalten Sie eine vollständige Beschreibung, wie Adressen eingegeben werden können.

Der Befehl *Previous* bringt den Code-Ausschnitt wieder dorthin zurück, wo er vor dem Aufruf eines *Goto*-Befehls gewesen ist.

Origin

Mit dem Befehl *Origin* (engl. Ursprung) kommt man zur aktuellen Programmposition, wie Sie im Registerpaar CS:IP steht. Dieser Befehl ist sehr praktisch, um wieder zum Ausgangspunkt zurückzukehren.

Der Befehl *Previous* bringt den Code-Ausschnitt wieder dorthin zurück, wo er vor dem Aufruf eines Origin-Befehls gewesen ist.

Follow

Der Cursor wechselt beim Befehl *Follow* (engl. [ver]folgen) zur Zieladresse des hervorgehobenen Befehls. Der Code-Ausschnitt steht an der Stelle im Programmcode, die im hervorgehobenen Befehl adressiert wird. Bei bedingten Sprungbefehlen wird davon ausgegangen, daß die Verzweigung ausgeführt wird. Dieser Befehl kann im Zusammenhang mit allen CALL-Befehlen, JMP-Befehlen, den bedingten Verzweigungen (`JNZ`, `JO`, `LOOP`, `JCXZ` usw.) und den `INT`-Befehlen verwendet werden.

Mit dem Befehl *Previous* wird der Codeausschnitt wieder an die Position zurückgesetzt, die vor dem *Follow*-Befehl angezeigt wurde.

Caller

Der Codeausschnitt wechselt beim Aufruf von *Caller* zu dem Befehl, der die aktuelle Routine aufgerufen oder den Interrupt ausgelöst hat. Der Befehl funktioniert nicht immer. Wenn die Interrupt-Routinen oder das Unterprogramm selbst einige Werte auf den Stack gelegt haben, kann Turbo Debugger nicht herausfinden, woher der Aufruf kam.

Mit dem Befehl *Previous* wird der Codeausschnitt wieder an die Position zurückgesetzt, die vor dem Befehl *Caller* angezeigt wurde.

Previous

Mit dem Befehl *Previous* (engl. vorherige [Position]) wird der Code-Ausschnitt wieder an die Adresse zurückgebracht, die der Code-Ausschnitt vor dem letzten Aufruf einer ausdrücklichen Adreßänderung inne hatte. Wenn man den Code-

Ausschnitt mit den Pfeiltasten oder Bild↑ und Bild↓ verändert, wird die Position nicht festgehalten. Mit *Previous* kommt man dann nicht zurück.

Wenn Sie den Befehl *Previous* auswählen, dann merkt sich Turbo Debugger die Position des Code-Ausschnitts wieder, so daß mit wiederholten *Previous*-Befehlen zwischen zwei Adressen umgeschaltet werden kann.

Search

Man kann mit *Search* (engl. suchen) einen Befehl oder eine Byte-Folge zum Suchen eingeben. Die Eingabe der Daten erfolgt wie beim Befehl *Assemble*.

Mit den gesuchten Befehlen sollten Sie vorsichtig sein. Sie sollten nur nach Befehlen suchen, die die gesuchten Bytes nicht verändern, wenn sie assembliert werden. Die Suche nach den folgenden Befehlen ist kein Problem:

```
push   dx
pop    [di+4]
add    ax,100
```

Wenn Sie aber die folgenden Befehle suchen, kann es zu unvorhersehbaren Ergebnissen kommen:

```
je     123
call   funker
loop   100
```

Anstelle eines Befehls kann man auch eine Byte-Folge eingeben. In Kapitel 9, »Ausdrücke«, erhalten Sie ausführliche Informationen über die Möglichkeiten der Eingabe von Byte-Folgen.

View Source

Dieser Befehl öffnet ein *Module*-Fenster mit dem Quellcode des gerade disassemblierten Codes. Wenn kein entsprechender Quellcode vorliegt (zum Beispiel in Windows-Code, oder wenn keine Debug-Informationen vorliegen), bleibt der Code-Ausschnitt aktiv.

Mixed

Der Befehl *Mixed* (engl. gemischte Darstellung) schaltet zwischen den drei möglichen Darstellungen von disassemblierten Befehlen und dem Quellcode um:

No — Es wird überhaupt kein Quellcode dargestellt, sondern nur die disassemblierten Befehle.

Yes — Die Zeilen des Quellcodes erscheinen vor dem ersten disassemblierten Befehl dieser Quellzeile. Der Ausschnitt wird mit dieser Darstellungsart aktiviert, wenn das aktuelle Modul in einer höheren Programmiersprache geschrieben ist.

Both — Die Quellzeilen ersetzen die disassemblierten Zeilen mit dem entsprechenden Quellcode. Sonst erscheinen die disassemblierten Befehle. Dieser Modus ist anzuwenden, wenn man das Original im Quellcode sehen will und nicht den ent-

sprechenden disassemblierten Befehl. Der Ausschnitt wird automatisch mit dieser Darstellungsart aufgerufen, wenn der Quelltext des aktuellen Moduls in Assembler geschrieben ist. In diesem Fall ergibt sich auch der größte Vorteil. In C oder Pascal bringt diese Darstellungsmethode wenig, da lediglich die Anweisungen mit Registervariablen direkt umgesetzt werden, alle anderen Anweisungen werden disassembliert.

New CS:IP

Die aktuelle Programmposition (CS:IP) wird mit *New CS:IP* auf die hervorgehobene Adresse versetzt. Wenn Sie Ihr Programm erneut laufen lassen, beginnt die Ausführung an dieser Adresse. Das ist sehr praktisch, um einen Code-Ausschnitt, der nicht ausgeführt werden soll, zu überspringen.

Achtung: Verwenden Sie diesen Befehl nur mit äußerster Vorsicht. Wenn Sie CS:IP an eine Stelle setzen, in der der Stack in einem anderen Zustand sein sollte als an der aktuellen Programmposition in CS:IP, dann kommt es mit fast hundertprozentiger Sicherheit zu einem Systemabsturz. Sie sollten diesen Befehl nicht dazu verwenden, um CS:IP an eine Adresse außerhalb der aktuellen Routine zu versetzen.

Assemble

Mit dem Aufruf von *Assemble* (engl. assemblieren) kann man einen Befehl assemblieren, der den Befehl an der momentan hervorgehobenen Stelle im Programm temporär ersetzt. Sie werden sofort nach dem Befehl gefragt, der assembliert werden soll. Weitere Einzelheiten finden Sie im Abschnitt »Der Assembler« im weiteren Verlauf dieses Kapitels.

Sie können diesen Befehl auch durch einfaches Eingeben des Assemblerbefehls aktivieren. Sie erhalten dann sofort ein Dialogfenster wie beim Befehl *Assemble*.

I/O

Im Menü I/O (engl. Abk. Ein-/Ausgabe) kann man einen Wert in den I/O-Adreßraum der CPU schreiben oder von dort auslesen. Damit kann auch der Inhalt von Ein-/Ausgaberegistern auf Zusatzplatinen gelesen und verändert werden. Nach der Eingabe des Befehls stehen vier Optionen bereit:

In byte (engl. Byte ein[lesen]) liest ein Byte von einer Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie ermitteln wollen. Mit Hilfe von *Read word* können Sie ein Wort aus einer Ein/Ausgabeadresse einlesen.

Out byte (engl. Byte aus[geben]) schreibt ein Byte in eine Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie festlegen wollen. Mit Hilfe von *Write word* können Sie ein Wort in eine Ein/Ausgabeadresse schreiben.

Read word (engl. Wort lesen) liest ein Wort von einer Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse ge-

fragt, deren Wert Sie ermitteln wollen. Mit Hilfe von *In byte* können Sie ein Byte aus einer Ein/Ausgabeadresse einlesen.

Write word (engl. Wort schreiben) schreibt ein Wort in eine Ein-/Ausgabeadresse. In einem Dialogfenster werden Sie nach der Ein-/Ausgabeadresse gefragt, deren Wert Sie festlegen wollen. Mit Hilfe von *Out byte* können Sie ein Byte in eine Ein/Ausgabeadresse schreiben. IN- und OUT-Befehle greifen auf den Ein-/Ausgabebereich zu, in dem sich unter anderem Geräte wie serielle Schnittstellen, Festplatten-Controller und Bildschirmadapter befinden.

Achtung: Mit diesen Befehlen sollten Sie vorsichtig sein. Einige Ein-/Ausgabegeräte betrachten das Einlesen eines Ports als einen Befehlszugriff, bei dem zum Beispiel Statusbits zurückgesetzt oder neue Daten eingelesen werden. Es kann sein, daß dadurch der normale Ablauf Ihres Programms nicht mehr gewährleistet ist.

Die Register- und Statusausschnitte

Der Registerausschnitt ist der obere Ausschnitt rechts vom Code-Ausschnitt. Er zeigt den Inhalt der CPU-Register.

Der rechte obere Ausschnitt ist der Statusausschnitt, in dem die Einstellung der acht CPU-Flags gezeigt wird. In der folgenden Tabelle sind die verschiedenen Flags mit ihrer Darstellung im Statusausschnitt aufgelistet:

Flag	Bezeichnung des Flags
c	Carry
z	Zero
s	Vorzeichen (Sign)
o	Überlauf (Overflow)
p	Parität
a	Hilfscarry (Auxiliary)
i	Interrupt
d	Direction

Das lokale Menü des Registerausschnitts

Mit Alt F10 erhalten Sie das lokale Menü des Registerausschnitts. Wenn die Strg -Tastenkürzel aktiviert sind, können Sie die einzelnen Menübefehle aber auch mit der Taste Strg und dem ersten Buchstaben des Befehls direkt aufrufen.

Increment
Der Befehl *Increment* (engl. erhöhen) erhöht den Wert des hervorgehobenen Registers um Eins. Dies stellt eine einfache Methode dar, um durch geringfügige Änderungen an Registerwerten falsche Indexgrenzen auszugleichen.

Decrement
Der Befehl *Decrement* (engl. erniedrigen) erniedrigt das gerade hervorgehobene Register um Eins.

Zero
Der Befehl *Zero* (engl. Null) setzt das hervorgehobene Register auf Null.

Change...
Mit dem Befehl *Change* (engl. verändern) kann der Wert des hervorgehobenen Registers verändert werden. Nach der Auswahl dieses Befehls werden Sie um den neuen Wert gebeten. Bei der Eingabe des Werts können Sie alle Möglichkeiten des Syntaxparsers voll ausschöpfen.

Sie können diesen Befehl auch aktivieren, indem Sie einfach mit der Eingabe des neuen Registerwerts anfangen. Wenn Sie das machen, erscheint sofort ein Dialogfenster mit einem Eingabefeld wie bei der Eingabe des Befehls *Change*.

Registers 32-bit
Der Befehl *Registers 32-bit* schaltet die Darstellung bei einem 80386- oder nachfolgenden Prozessoren zwischen 16- und 32-Bit-Registerwerten um. Normalerweise erscheinen 16-Bit-Register, falls Sie diesen Befehl nicht verwenden, um die Darstellung auf 32-Bit umzustellen. Sie brauchen die 32-Bit-Register nur, wenn Sie ein Programm debuggen, das tatsächlich die 32-Bit-Adressierung des 80386-Chips ausnützt. Wenn Sie ein gewöhnliches Programm mit der normalen 16-Bit-Adressierung debuggen, können Sie auf die übersichtlichere 16-Bit-Darstellung der Register umschalten.

Das lokale Menü des Statusausschnitts

Mit Alt F10 erhalten Sie das lokale Menü des Statusausschnitts. Sie können die einzelnen Menübefehle aber auch mit der Taste Strg und dem ersten Buchstaben des Befehls aufrufen, wenn die Strg -Tastenkürzel aktiviert sind.

Toggle
Setzt den Wert des Flags auf 0, wenn er vorher 1 war, und auf 1, wenn er vorher 0 war. Der Wert 0 entspricht einem »gelöschten« Bit und 1 einem »gesetzten« Bit. Sie können auch einfach CR eingeben, um den Wert des hervorgehobenen Flags umzuschalten.

Der Selector-Ausschnitt

Dieser Ausschnitt zeigt eine Liste der Selektoren im Protected-Modus mit einigen zusätzlichen Informationen.

Ein Selektor kann entweder gültig oder ungültig sein. Falls ein Selektor gültig ist, zeigt er auch eine Position in der Deskriptortabelle, die einer Speicheradresse entspricht. Wenn der Selektor ungültig ist, wird er momentan nicht verwendet.

Bei einem gültigen Selektor werden folgende Informationen im Ausschnitt angezeigt:

- ob es sich beim Inhalt um Daten oder Code handelt;
- ob der vom Selektor adressierte Speicherbereich geladen ist (im Speicher vorliegt) oder nicht geladen ist (auf Platte ausgelagert);
- die Länge des Speichersegments in Bytes.

Wenn der Selector ein Datensegment adressiert, werden weitere Informationen über die Zugriffsrechte (Lesen/Schreiben) und die Ausdehnungsrichtung des Segments (nach oben oder unten) angegeben.

Das lokale Menü des Selector-Ausschnitts

Wenn Sie mit dem Cursor im *Selector*-Ausschnitt sind, können Sie mit Alt F10 das lokale Menü aufrufen. Sie können die einzelnen Menübefehle aber auch mit der Taste Strg und dem ersten Buchstaben des Befehls direkt aufrufen, wenn die Strg -Tastenkürzel aktiviert sind.

Mit dem lokalen Menü des *Selector*-Ausschnitts kann man einen neuen Selektor auswählen (der Befehl *Selector*) oder den Inhalt des aktuellen Selektors anzeigen (der Befehl *Examine*). Der Inhalt wird entweder im Code- oder im Datenausschnitt angezeigt, je nachdem, um welche Art von Selektor es sich handelt.

Selector

Bei diesem Befehl geben Sie den Selektor an, der angezeigt werden soll. Sie können alle Syntaxoptionen verwenden, um den Selektor zu spezifizieren. Wenn Sie einen numerischen Wert eingeben, geht TDW von einer Dezimalzahl aus, falls Sie nicht die Syntax der aktuellen Sprache für eine Hexadezimalzahl verwenden.

Wenn zum Beispiel C als aktuelle Sprache eingestellt ist, können Sie den Hexadezimalwert 7F eines Selektors mit `0x7F` eingeben. In Pascal würden Sie `$7F` verwenden. Sie könnten jedoch auch den dezimalen Wert 127 verwenden, um den Selektor 7F anzugeben.

Eine andere Methode für die Auswahl eines Selektors arbeitet mit den im CPU-Fenster angezeigten Segmentregistern. Wenn ein Register einen Selektor enthält, für den Sie sich interessieren, können Sie auch den Namen des entspre-

chenden Registers mit einem vorangestellten Unterstrich (_) verwenden. Den Selektor des Datensegments erhalten Sie zum Beispiel mit _DS.

Examine

Mit diesem Befehl können Sie sich den Inhalt des vom Selektor adressierten Speicherbereichs auzeigen lassen. Dabei wird automatisch auf den entsprechenden Ausschnitt umgeschaltet, in dem der Inhalt angezeigt wird. Wenn sich der Selektor auf ein Code-Segment bezieht, wird der Inhalt im Code-Ausschnitt wiedergegeben. Wenn sich der Selektor auf ein Datensegment bezieht, wird der Inhalt im Daten-Ausschnitt wiedergegeben.

Der Datenausschnitt

Dieser Ausschnitt bietet eine ungefilterte Darstellung eines ausgewählten Speicherbereichs. Im linken Bereich wird die Adresse der in dieser Zeile dargestellten Daten angegeben. Die Adresse wird entweder in Form eines hexadezimal angegebenen Segments und Offsets dargestellt, oder der Segmentwert wird durch die Bezeichnung des DS-Registers ersetzt, wenn der Wert des Segments mit dem aktuellen DS-Segment übereinstimmt. Der Datenausschnitt vergleicht die Segmentregister in folgender Reihenfolge: DS, ES, SS und CS.

Dahinter werden die Daten von einem oder mehreren Datenobjekten dargestellt. Das Format dieses Bereichs hängt von der Darstellungsart ab, die mit dem Menübefehl *Display as* ausgewählt wurde. Wenn Sie eines der Fließkommaformate (*Comp*, *Float*, *Real*, *Double*, *Extended*) eingestellt haben, wird nur eine einzige Fließkommazahl in einer Zeile dargestellt. Beim *Byte*-Format werden acht Bytes in einer Zeile dargestellt, beim Format *Word* werden vier Worte pro Zeile dargestellt und beim *Long*-Format zwei Doppelworte.

Im rechten Teil der Darstellung werden die Zeichen gezeigt, die den Bytes entsprechen. Turbo Debugger gibt zu allen Byte-Werten das entsprechende Bildschirmzeichen aus. Sie brauchen deshalb nicht überrascht zu sein, wenn Sie seltsame Zeichen auf der rechten Seite des Speicher-Dumps sehen. Es sind dies lediglich die entsprechenden Bildschirmzeichen für die nebenstehenden Hexadezimalwerte.

Die Anzahl der Bytes, die in einer Zeile dargestellt werden, hängt vom Format ab, das mit dem Befehl *Display as* eingestellt wurde.

Achtung: Wenn Sie den Datenausschnitt zur Untersuchung des Bildschirmspeichers, der ROM-BIOS-Daten oder der Interrupt-Vektoren verwenden, dann sehen Sie die Werte des Programms, das untersucht wird, und nicht die tatsächlich vorhandenen Werte, wenn Turbo Debugger läuft. Dies sind nicht immer dieselben Werte, die zu dem Zeitpunkt, da Sie sie untersuchen, wirklich im Speicher stehen. Turbo Debugger merkt sofort, wenn Sie auf Speicherbereiche zugreifen, die er auch verwendet, und liefert Ihnen die richtigen Werte aus den Bereichen, in denen die Werte Ihres Programms gespeichert sind.

Das lokale Menü des Datenausschnitts

Wenn Sie mit dem Cursor im Datenausschnitt sind, können Sie mit Alt F10 das lokale Menü aufrufen. Sie können die einzelnen Menübefehle aber auch mit der Taste Strg und dem ersten Buchstaben des Befehls direkt aufrufen, wenn die Strg -Tastenkürzel aktiviert sind.

Goto
Mit dem Befehl *Goto* (engl. gehe zu) kommen Sie an eine bestimmte Adresse in Ihren Daten. Geben Sie die neue Adresse ein, die Sie sehen wollen. Sie können auch Adressen eingeben, die außerhalb Ihres Programms liegen und zum Beispiel von DOS, von speicherresidenten Programmen oder vom BIOS verwendet werden. In Kapitel 9, »Ausdrücke«, wird ausführlich erläutert, welche Möglichkeiten für die Eingabe von Adressen zur Verfügung stehen.

Search
Der Befehl *Search* (engl. suchen) sucht einen String ab der aktuellen Speicheradresse. Geben Sie anschließend die gesuchte Byte-Folge ein. Wenn man zum Ende eines Segments kommt, wird die Suche nicht am Anfang des Segments fortgesetzt. In Kapitel 9, »Ausdrücke«, finden Sie eine vollständige Erklärung der Syntax für Byte-Folgen.

Next
Mit Hilfe des Befehls *Next* (engl. Nächster) wird die nächste Position einer Byte-Folge gesucht, die Sie in einem *Search* -Befehl zuvor schon gesucht haben.

Change
Der Befehl *Change* ermöglicht die Veränderung von Bytes an der aktuellen Cursorposition. Wenn Sie in der ASCII-Darstellung sind oder als Format *Byte* ausgewählt wurde, dann werden Sie nach einer Byte-Folge gefragt. Ansonsten müssen Sie einen Wert für den aktuell dargestellten Datentyp eingeben. In Kapitel 9, »Ausdrücke«, werden die Byte-Folgen ausführlich beschrieben.
Sie können diesen Befehl auch einfach dadurch aktivieren, daß Sie anfangen, den neuen Wert einzugeben. Daraufhin erhalten Sie ein Eingabefeld in einem Dialogfenster wie beim Befehl *Change* .

Follow
Durch die Auswahl des Befehls *Follow* (engl. [ver]folgen) erhalten Sie ein Menü, in dem Sie NEAR- oder FAR-Zeigerketten zurückverfolgen können.

Near code interpretiert das Wort unter dem Cursor im Datenausschnitt als einen Offset im aktuellen Code-Segment, das im CS-Register steht. Der Code-Ausschnitt wird zum aktuellen Ausschnitt und zeigt die ausgewählte Adresse an.

Far code interpretiert das Doppelwort unter dem Cursor im Datenausschnitt als eine FAR-Adresse (Segment und Offset). Der Code-Ausschnitt wird zum aktuellen Ausschnitt und zeigt die ausgewählte Adresse an.

Offset to data erlaubt die Verfolgung von Zeigerketten, die aus einem Wort (NEAR, nur Offset) bestehen. Der Datenausschnitt wird auf den Offset positioniert, der im Wort in der Speicherstelle des Cursors steht.

Segment:offset to data erlaubt die Verfolgung von Zeigerketten, die aus einem Doppelwort (FAR, Segment und Offset) bestehen. Der Datenausschnitt wird auf den Offset positioniert, der in den beiden Worten in der Speicherstelle des Cursors steht.

Base segment:0 to data wertet das Wort unter dem Cursor als eine Segmentadresse aus und positioniert den Datenausschnitt an den Anfang dieses Segments.

Previous

Der Befehl *Previous* (engl. vorherige [Position]) positioniert den Datenausschnitt an die Stelle, die der Datenausschnitt vor der letzten ausdrücklichen Veränderung innehatte. Bei Veränderungen mit den Pfeiltasten oder Bild↑ und Bild↓ wird die letzte Position nicht festgehalten.

Turbo Debugger merkt sich die letzte Position auf einem Stack mit den letzten fünf Adressen, so daß man mehrere Aufrufe der Befehle des Menüs *Follow* oder von *Goto* zurückverfolgen kann.

Display as

Mit Hilfe des Befehls *Display as* (engl. Darstellung als) kann die Erscheinungsweise der Daten im Datenausschnitt festgelegt werden. Dabei kann man alle Formate verwenden, die von C, Pascal oder Assembler unterstützt werden. Sie können eine der nachfolgend beschriebenen Einstellungen auswählen.

Byte stellt den Datenausschnitt auf die Darstellung hexadezimaler Bytes um. Dies entspricht dem Datentyp `char` in C und dem Datentyp `byte` in Pascal.

Word sorgt dafür, daß die Daten im Datenausschnitt in Form von hexadezimalen Worten ausgegeben werden. Dies entspricht dem Datentyp `int` in C und dem Datentyp `word` in Pascal.

Long stellt den Datenausschnitt auf die Darstellung von hexadezimalen Doppelworten um. Dies entspricht dem Datentyp `long` in C und dem Datentyp `longint` in Pascal.

Comp stellt den Datenausschnitt auf die Ausgabe von 8-Byte-Integerzahlen um. Der Wert wird dezimal angezeigt. Dies entspricht dem Datentyp `comp` (IEEE) in Pascal.

Float stellt den Datenausschnitt auf die Darstellung von Fließkommazahlen einfacher Genauigkeit ein. Die Werte werden in Exponentialdarstellung ausgegeben. Dies entspricht dem Datentyp `float` in C und dem Datentyp `single` (IEEE) in Pascal.

Real stellt den Datenausschnitt auf die Darstellung von Pascal-Fließkommazahlen (sechs Bytes) um. Die Werte werden in Exponentialdarstellung ausgegeben. Dies entspricht dem Datentyp `real` in Pascal.

Double stellt den Datenausschnitt auf die Darstellung von Fließkommazahlen doppelter Genauigkeit (acht Bytes) um. Die Werte erscheinen in Exponentialdarstellung. Dies entspricht dem Datentyp `double` in C und dem Datentyp `double` (IEEE) in Pascal.

Extended stellt die Ausgabe des Datenausschnitts auf Fließkommazahlen (10 Bytes) erweiterter Genauigkeit um. Die Werte erscheinen in Exponentialdarstellung. Dies entspricht dem Datentyp `long double` in C und dem Datentyp `extended` (IEEE) in Pascal.

Block
In Menü *Block* lassen sich ganze Speicherblöcke manipulieren. Sie können Speicherbereiche kopieren, löschen und initialisieren, sowie von Dateien lesen und in Dateien schreiben. Mit dem Befehl *Block* erhalten Sie die nachstehende Optionen.

Clear setzt einen zusammenhängenden Speicherbereich auf Null (0). Sie werden nach der Adresse und der Anzahl der Bytes gefragt, die gelöscht werden sollen.

Move kopiert einen Speicherblock von einer Adresse an eine andere. Sie werden nach der Quelladresse, der Zieladresse und der Anzahl der Bytes gefragt.

Set initialisiert einen zusammenhängenden Speicherbereich mit einem bestimmten Wert. Sie werden nach der Adresse und der Anzahl der Bytes gefragt, sowie nach dem Wert, auf den sie gesetzt werden sollen.

Read liest den Inhalt von Dateien ganz oder teilweise in den Speicher ein. Sie müssen zunächst den Namen der Datei, anschließend die Speicheradresse und dann die Anzahl der Bytes angeben.

Write schreibt einen Speicherbereich in eine Datei. Sie müssen zunächst den Namen der Datei, anschließend die Speicheradresse und dann die Anzahl der Bytes angeben.

Der Stack-Ausschnitt

In der rechten unteren Ecke des *CPU*-Fensters befindet sich der Stackausschnitt, in dem der Inhalt des Stacks angezeigt wird. (Ein Pfeil gibt die aktuelle Position des Stackzeigers SS:SP wieder.)

Das lokale Menü des Stack-Ausschnitts

Mit Alt F10 erhalten Sie das lokale Menü des Stack-Ausschnitts. Wenn die Strg -Tastenkürzel aktiviert sind, können Sie die einzelnen Menübefehle mit der Taste Strg und dem ersten Buchstaben des Befehls auch direkt aufrufen.

Goto

Mit dem Befehl *Goto* (engl. gehe zu) können Sie an eine bestimmte Stelle auf dem Stack springen. Geben Sie anschließend die neue Stackadresse ein. Wenn Sie wollen, können Sie auch Adressen außerhalb Ihres Programmstacks angeben, obwohl Sie für diese Zwecke auch den Datenausschnitt zur Verfügung haben, in dem Sie beliebige Speicherbereiche darstellen können. In Kapitel 9, »Ausdrükke«, finden Sie eine Beschreibung, auf welche Art und Weise die Adressen eingegeben werden können.

Der Befehl *Previous* bringt den Stackausschnitt wieder an die Position zurück, an der der Stack vor dem Aufruf des letzten *Goto*-Befehls war.

Origin

Durch die Eingabe des Befehls *Origin* (engl. Ursprung) wird der Cursor wieder an die von SS:SP adressierte Speicherstelle zurückversetzt. Dieser Befehl ist sehr praktisch, wenn man wieder zu seinem Ausgangspunkt zurückkehren will.

Der Befehl *Previous* bringt den Stackausschnitt wieder an die Position zurück, an der der Stack vor dem Aufruf des letzten *Origin*-Befehls war.

Follow

Mit Hilfe des Befehls *Follow* (engl. [ver]folgen) werden Sie an die Stelle im Stack positioniert, die im gerade hervorgehobenen Wort adressiert wird. Diese Option eignet sich vor allem zum Zurückverfolgen zur aufrufenden Funktion über die Parameter am Stack.

Der Befehl *Previous* bringt den Stackausschnitt wieder an die Position, an der der Stack vor dem Aufruf des letzten *Follow*-Befehls war.

Previous

Der Befehl *Previous* (engl. vorherige [Position]) positioniert den Stackausschnitt an die Stelle, die der Stackausschnitt vor der letzten ausdrücklichen Veränderung des Stacks innehatte. Bei Veränderungen mit den Pfeiltasten oder Bild↑ und Bild↓ wird die letzte Position nicht festgehalten.

Wenn der Befehl *Previous* mehrmals hintereinander eingegeben wird, springt der Stack-Ausschnitt zwischen zwei Adressen hin und her.

Change...

Mit dem Befehl *Change* (engl. verändern) kann man in der hervorgehobenen Speicherstelle auf dem Stack einen neuen Wert eingeben.

Sie können diesen Befehl auch aktivieren, indem Sie einfach mit der Eingabe des neuen Registerwerts anfangen. Wenn Sie das machen, erscheint sofort ein Dialogfenster, als ob Sie den Befehl *Change* eingegeben hätten.

Der Assembler

Mit dem Befehl *Assemble* im lokalen Menü des Code-Ausschnitts kann man Befehle für die Prozessoren 8086, 80186, 80286, 80386 und i486 sowie die Koprozessoren 8087, 80287 und 80387 assemblieren.

Wenn Sie den in Turbo Debugger integrierten Assembler verwenden, um Ihr Programm zu verändern, werden diese Änderungen nicht dauerhaft vorgenommen. Wenn Sie Ihr Programm mit dem Befehl *Run/Program reset* erneut laden oder ein anderes Programm mit dem Befehl *File/Open* laden, sind alle von Ihnen durchgeführten Veränderungen wieder verloren.

Üblicherweise verwendet man den Assembler, um eine Idee für eine bestimmte Problemlösung in einem Programm schnell auszuprobieren. Sobald Sie nachgeprüft haben, daß die Veränderung funktioniert, müssen Sie den Debugger verlassen, in Ihrem Quellcode alles nachtragen und anschließend Ihr Programm neu übersetzen und linken.

Der folgende Abschnitt beschreibt die Unterschiede zwischen dem integrierten Assembler und der Syntax, die von Turbo Assembler akzeptiert wird.

Vorgaben für die Größe von Adreßoperanden

Bei den Unterprogrammaufrufen (`CALL`), Sprungbefehlen (`JMP`) und den bedingten Verzweigungen (`JNE`, `JL` usw.) erzeugt der Assembler automatisch immer den kürzesten Befehl für die gegebene Entfernung. Mit Hilfe der Operandenvorgaben `NEAR` und `FAR` vor der Zieladresse läßt sich der Befehl mit einer bestimmten Größe assemblieren. Zum Beispiel:

```
CALL    FAR XYZ
JMP     NEAR A1
```

Konstanten und Speicheroperanden

Wenn Sie ein Symbol aus Ihrem Programm als Befehlsoperanden verwenden, müssen Sie dem integrierten Assembler mitteilen, ob Sie den Inhalt des Symbols oder die Adresse des Symbols wollen. Wenn Sie nur den Namen des Symbols

angeben oder das Symbol in eckige Klammern ([ ]) schreiben, wird es zu einer Speicherreferenz. Wenn Ihr Programm zum Beispiel die Datendefinition

```
a       DW 4
```

enthält und Sie einen Befehl assemblieren oder einen Ausdruck auswerten wollen, der sich auf den Wert dieser Variablen bezieht, so benutzen Sie den Namen der Variablen allein oder in Klammern:

```
mov     ax,[a]
mov     dx,a
```

Um auf die Adresse der Variablen zuzugreifen, benutzt man den Operator OFFSET:

```
mov     ax,OFFSET a
```

Vorgaben für den Datentyp eines Operanden

Bei einigen Befehlen muß man die Operandengröße mit Hilfe eines der beiden folgenden Ausdrücke vor dem Operanden angeben:

```
BYTE    PTR
WORD    PTR
```

Zwei Beispiele für Befehle, in denen diese Vorgaben enthalten sind:

```
add     BYTE PTR [si],10
mov     WORD PTR [bp+10],99
```

Außer diesen beiden Datentypvorgaben kann man noch die folgenden Vorgaben verwenden, wenn man Befehle für den 80x87-Koprozessor assemblieren will:

```
DWORD   PTR
QWORD   PTR
TBYTE   PTR
```

Zwei Beispiele für diese Vorgaben:

```
fild    QWORD PTR [bx]
fstp    TBYTE PTR [bp+4]
```

String-befehle

Wenn Sie einen Stringbefehl assemblieren, dann müssen Sie die Größe (Byte oder Word) in der Kurzform des Befehls angeben. Der Assembler akzeptiert keine Stringbefehle, in denen bei der Kurzform nur die Operanden mit ihren Datentypen stehen, bei denen aber im Befehl keine Angabe zur Datengröße enthalten ist. Sie müssen zum Beispiel STOSW anstelle von STOS WORD PTR[DI] hinschreiben.

Das Fenster Dump

Das *Dump*-Fenster zeigt Ihnen den reinen Auszug eines beliebigen Speicherbereichs. Es funktioniert genauso wie der Datenausschnitt im *CPU*-Fenster, mit dem Unterschied, daß bei voller Größe des Fensters doppelt soviele Daten in einer Zeile angezeigt werden.

Im vorangegangenen Abschnitt »Das lokale Menü des Datenausschnitts« dieses Kapitels finden Sie eine Beschreibung des Inhalts und der lokalen Menüs für dieses Fenster.

Normalerweise verwendet man dieses Fenster, wenn man ein Assemblerprogramm auf Quellcode-Ebene untersucht und einen direkten Blick auf manche Datenbereiche werfen will. Mit dem Befehl *View/Dump* oder *View/Another/Dump* können Sie ein *Dump*-Fenster anlegen und mehrere Bereiche untersuchen.

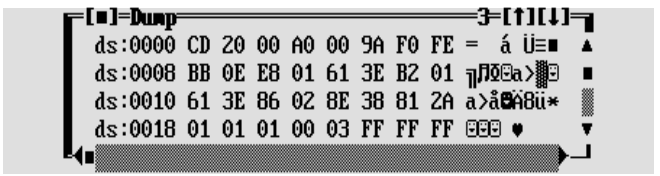
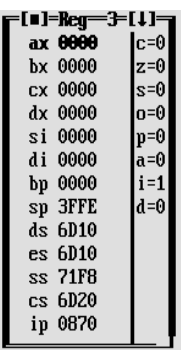


Abbildung 12.2 Das Dump-Fenster

Sie können dieses Fenster auch in einem *Inspector*-Fenster verwenden, wenn Sie einen Blick auf die reinen Bytes Ihrer untersuchten Datenobjekte werfen wollen. Verwenden Sie den Befehl *View/Dump*, um ein *Dump*-Fenster zu erhalten, das die im *Inspector*-Fenster stehenden Daten zeigt.

Das Fenster Registers

Das Fenster *Registers* zeigt den Inhalt der CPU-Register und der Flags. Es ist eine Kombination der Register- und Statusausschnitte des *CPU*-Fensters.



Abbild 12.3 Das Fenster Registers

In den vorangegangenen Abschnitten »Das lokale Menü des Registerausschnitts« und »Das lokale Menü des Statusausschnitts« dieses Kapitels finden Sie eine Beschreibung des Inhalts und der lokalen Menüs in diesem Fenster.

Verwenden Sie dieses Fenster, wenn Sie ein Assemblerprogramm auf Quellcode-Ebene debuggen und die Registerwerte sehen wollen.

13. 80x87-Koprozessor und Gleitkomma-Emulator

Wenn Sie in Ihrem Programm Gleitkommazahlen verwenden, dann können Sie mit TDW den Status des 80x87-Arithmetikprozessors bzw. des Emulators untersuchen und verändern. Dieses Kapitel richtet sich an Programmierer, die mit den Operationen des 80x87-Koprozessors bereits vertraut sind. Man muß die Möglichkeiten, die in diesem Kapitel zum Debuggen von Programmen mit Gleitkommazahlen beschrieben werden, nicht unbedingt ausnutzen, obwohl man einige vertrackte Fehler damit leichter finden kann.

In diesem Kapitel stellen wir die Unterschiede zwischen dem 80x87-Koprozessor und dem Software-Emulator heraus. Wir beschreiben das Fenster *Numeric Processor* und zeigen Ihnen, wie man die Gleitkommaregister, sowie die Status- und Control-Flags untersuchen und verändern kann.

Der 80x87-Koprozessor im Vergleich zum Emulator

TDW stellt automatisch fest, ob Ihr Programm den Arithmetikprozessor oder den Emulator verwendet, und paßt seine Arbeitsweise dementsprechend an.

Beachten Sie bitte, daß Programme die Gleitkommazahlen verarbeiten, entweder den Emulator oder den Koprozessor verwenden können – eine gleichzeitige Verwendung beider ist ausgeschlossen. Wenn Sie ein spezielles Assembler-Programm geschrieben haben, das beide verwendet, dann ist Turbo Debugger nicht in der Lage, den Status des Koprozessors darzustellen. Es kann dann lediglich der Zustand des Emulators wiedergegeben werden.

Das Fenster Numeric Processor

Von der Menüleiste aus öffnen Sie mit dem Befehl *View/Numeric processor* das Fenster *Numeric Processor*. Die oberste Zeile des Fensters enthält den aktuellen Programmzeiger (*IPTR*), den Zeiger auf Datenbereiche (*OPTR*) und den Befehl als Hexadezimalcode (*OPCODE*). Der Datenzeiger und der Programmzeiger werden als physikalische 20-Bit-Adressen angegeben. Sie können diese Adressen

in Segment:Offset-Adressen umwandeln, wenn Sie die ersten vier Ziffern als Segment auffassen und die letzte Ziffer als Offset verwenden.

Wenn in der obersten Zeile zum Beispiel `IPTR=5A669` steht, kann dieser Wert in die Adresse 5A66:9 zerlegt werden, falls Sie den aktuellen Befehl und die entsprechenden Daten im *CPU*-Fenster untersuchen wollen.

Das Fenster *Numeric Processor* gliedert sich in drei Abschnitte: Der linke Ausschnitt (Registerausschnitt) zeigt den Inhalt der Gleitkommaregister, der mittlere Ausschnitt (Control-Ausschnitt) enthält die Control-Flags und der rechte Ausschnitt (Statusausschnitt) zeigt die Status-Flags.

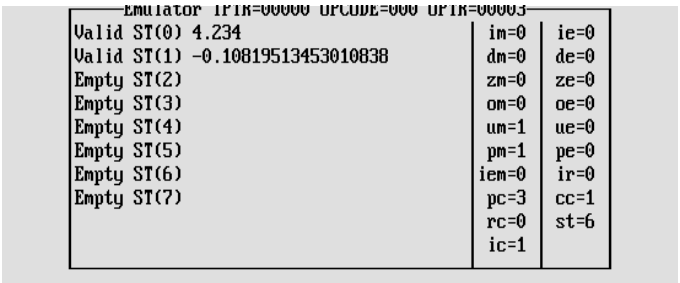


Abbildung 13.1 Das Fenster Numeric Processor

Die oberste Zeile enthält außerdem den letzten ausgeführten Gleitkommabefehl.

- *Emulator* gibt an, daß der Koprozessor emuliert wird. Falls ein Koprezessor vorliegt, wird entweder 8087, 80287 oder 80387 angezeigt.
- In IPTR steht die physikalische 20-Bit-Adresse der letzten Gleitkommaoperation.
- In OPCODE wird angegeben, welcher Befehl ausgeführt wurde.
- Hinter OPTR finden wir die Speicherstelle, die mit diesem Befehl adressiert wurde, falls ein Speicheroperand im Befehl vorkam.

Der Registerausschnitt

Die 80-Bit-Gleitkommaregister

Im Registerausschnitt sind die einzelnen Gleitkommaregister (ST(0) bis ST(7) zusammen mit ihrem Status dargestellt (*valid* [zulässig] / *zero* [Null] / *special* [Sonderfall] / *empty* [leer]). Der Inhalt wird als eine 80-Bit-Gleitkommazahl angegeben.

Wenn Sie das Fenster *Numeric Processor* auf den gesamten Bildschirm ausgedehnt haben (durch Eingabe von F5) oder mit dem Befehl *Window/ Move/Resize* verbreitert haben, dann sehen Sie die Gleitkommaregister auch in der reinen Hexadezimal-Darstellung.

Das lokale Menü des Registerausschnitts

Um das lokale Menü des Registerausschnitts zu erhalten, drücken Sie die Tastenkombination Alt F10 oder rufen den gewünschten Befehl direkt mit der Strg -Taste zusammen mit dem ersten Buchstaben des Menüpunkts auf:

Zero löscht das gerade hervorgehobene Register auf den Wert 0.

Empty setzt das hervorgehobene Register auf den Status *empty*. Dieser spezielle Status zeigt an, daß das Register keine gültigen Daten mehr enthält.

Change Mit diesem Befehl kann das hervorgehobene Register mit einem neuen Wert geladen werden. Nach dem Aufruf des Befehls werden Sie nach dem entsprechenden Zahlenwert gefragt. Sie können den Wert ganzzahlig oder als Gleitkommazahl eingeben, wobei der komplette Ausdrucks-Parser der eingestellten Programmiersprache zur Verfügung steht. Der eingegebene Wert wird automatisch in das 80-Bit-Gleitkommaformat des Koprozessors umgewandelt.

Der Befehl kann auch ganz einfach dadurch aktiviert werden, daß man einen neuen Wert für das Gleitkommaregister eingibt. Wenn Sie das machen, erscheint nach der ersten Eingabe sofort ein Dialogfenster, so als ob Sie den Befehl *Change* eingegeben hätten.

Der Control-ausschnitt

Die Control-Flags

Die folgende Tabelle enthält die verschiedenen Controll-Flags und ihre Darstellung im Control-Ausschnitt:

Bezeichnung im Ausschnitt	Beschreibung des Flags
im	Bit-Maske für unzulässige Operationen
dm	Bit-Maske für denormalisierte Operanden
zm	Bit-Maske für Division durch Null
om	Bit-Maske für Überlauf
um	Bit-Maske für Unterlauf
pm	Bit-Maske für Genauigkeitsverluste
iem	Bit-Maske für Interrupts (nur 8087)
pc	Genauigkeitskontrolle
rc	Rundungskontrolle
ic	Unendlichkeitsdarstellung

Das lokale Menü des Control-Ausschnitts

Mit Tab aktivieren Sie den Control-Ausschnitt. Drücken Sie dann Alt F10 , um das lokale Menü zu aktivieren. (Sie können auch die Strg -Taste mit dem ersten Buchstaben des Befehls verwenden, um den Befehl direkt aufzurufen.)

Toggle wandert durch die Werte, die man im gerade hervorgehobenen Control-Flag einstellen kann. Die meisten Flags können nur gesetzt oder gelöscht werden (0 oder 1), so daß der Wert des Flags lediglich umgeschaltet wird. Einige andere Flags können mehr als zwei Werte annehmen. Bei diesen Flags wird der Wert des Flags solange inkrementiert, bis der Maximalwert erreicht wird. Anschließend wird wieder bei 0 angefangen.

Die Control-Flags können auch mit der Taste CR umgeschaltet werden.

Der Status-ausschnitt

Die Status-Flags

Die folgende Tabelle enthält die verschiedenen Status-Flags und ihre Darstellung im Statusauschnitt:

Bezeichnung im Ausschnitt	Beschreibung des Flags
ie	Unzulässige Operation
de	Denormalisierter Operand
ze	Division durch Null
oe	Überlauf
ue	Unterlauf
po	Genauigkeitsverlust
ir	Interrupt-Anfrage
cc	Bedingungscode
st	Zeiger auf Stack-Spitze

Das lokale Menü des Statusausschnitts

Mit Tab kommen Sie zum Statusausschnitt. Sie drücken dann Alt F10 , um das lokale Menü zu aktivieren. (Sie können auch die Strg -Taste zusammen mit dem ersten Buchstaben des gewüschten Befehls verwenden, um den Befehl direkt aufzurufen.)

Toggle wandert durch die Werte, die man im gerade hervorgehobenen Status-Flag einstellen kann. Die meisten Flags können nur gesetzt oder gelöscht werden (0 oder 1), so daß der Wert des Flags lediglich umgeschaltet wird. Einige andere Flags können mehr als zwei Werte annehmen. Bei diesen Flags wird der Wert des Flags solange inkrementiert, bis der Maximalwert erreicht ist. Anschließend wird wieder bei 0 angefangen.

Die Status-Flags können auch mit CR umgeschaltet werden.

14. Die Befehlsübersicht

In den vorangegangenen Kapiteln wurden alle Befehle erklärt. Hier folgt nun eine kurze Zusammenfassung dieser Befehle. In diesem Kapitel beschreiben wir

- die Befehle, die mit einer einzigen Funktionstaste oder einer anderen Tastenkombination ausgelöst werden;
- die Befehle in der Menüleiste und den lokalen Menüs der einzelnen Fenster;
- alle Tastenkombinationen, die in den beiden Arten von Fensterausschnitten vorkommen (Eingabefelder und Listen);
- die Befehle zum Ausrichten der Größe und Position von Fenstern.

Funktionstasten und Tastenkürzel

Mit Funktionstasten und Tastenkürzel kann eine bestimmte Aktion unabhängig von der aktuellen Situation in der Oberfläche von Turbo Debugger ausgeführt werden. In Tabelle 14.1 ist die Bedeutung aller zulässigen Tastenkürzel und Funktionstasten aufgeführt.

Tabelle 14.1 Die Funktionstasten und Tastenkürzel

Taste	Menübefehl	Funktion
F1		Liefert kontextbezogene Hilfe stellungen.
F2	*Breakpoints / Toggle*	Breakpoint an der Cursor-Position setzen/löschen.
F3	*View / Module*	Auswahl eines Moduls.
F4	*Run / Go to cursor*	Programm bis zur Cursorposition laufen lassen.
F5	*Window / Zoom*	Aktuelles Fenster vergrößern/ verkleinern.
F6	*Window / Next Window*	Nächstes Fenster aktivieren.

Taste	Menübefehl	Funktion
F7	*Run / Trace into*	Einzelne Zeile oder Befehl ausführen.
F8	*Run / Step over*	Einzelne Zeile oder Befehl ausführen,, Unterprogrammaufruf überspringen.
F9	*Run / Run*	Programm ausführen.
F10		Menüleiste aktivieren.
Alt F1	*Help/Previous Topic*	Letzten Hilfebildschirm darstellen.
Alt F2	*Breakpoints/At*	Breakpoint auf eine bestimmte Adresse setzen.
Alt F3	*Window/Close*	Aktuelles Fenster schließen.
Alt F4	*Run/Back trace*	Programmausführung rückwärts.
Alt F5	*Window/User screen*	Aktuellen Ausgabebildschirm darstellen.
Alt F6	*Window/Undo close*	Zuletzt geschlossenes Fenster wieder öffnen.
Alt F7	*Run/Instruction trace*	Einzelbefehl ausführen.
Alt F8	*Run/Until return*	Programm bis zur Rückkehr aus einer Funktion ausführen.
Alt F9	*Run/Execute to*	Programm bis zu einer bestimmten Adresse ausführen.
Alt F10		Lokales Menü des aktuellen Fensters aktivieren.
Alt 1 - 9		Aktivieren eines der mit 1-9 numerierten Fenster.
Alt Leertaste		Aufruf des System-Menüs.
Alt B		Aufruf des *Breakpoints*-Menüs.
Alt D		Aufruf des *Data*-Menüs.
Alt E		Aufruf des *Edit*-Menüs
Alt F		Aufruf des *File*-Menüs.
Alt H		Aufruf des *Help*-Menüs.
Alt O		Aufruf des *Options*-Menüs.
Alt R		Aufruf des *Run*-Menüs.
Alt V		Aufruf des *View*-Menüs.
Alt W		Aufruf des *Window*-Menüs.
Alt X	*File/Quit*	Turbo Debugger verlassen und Rückkehr zu Windows.
Alt =	*Options/Macros/ Create*	*Tastaturmakro definieren.*
Alt -	*Options/Macros/ Stop recording*	*Makroaufnahme beenden.*

Taste	Menübefehl	Funktion
Strg F2	*Run/Program reset*	Programm erneut laden.
Strg F4	*Data/Evaluate*	Ausdruck auswerten.
Strg F5	*Window/Size/Move*	Position oder Größe eines Fensters ändern.
Strg F7	*Data/Watch*	Variable im *Watches*-Fenster aufnehmen.
Strg →		Verschiebt die Anfangsadresse in einem Code-Daten- oder Stack-Ausschnitt eines *CPU*-Fensters um ein Byte nach oben.
Strg ←		Verschiebt die Anfangsadresse in einem Code-Daten- oder Stack-Ausschnitt eines *CPU*-Fensters um ein Byte nach unten.
Umschalt F1	*Help/Index*	Aufruf des Hilfe-Index.
Umschalt F3	*Edit/Copy*	Kopie in Zwischenablage.
Umschalt F1	*Edit/Paste*	Einfügen aus Zwischenablage.
Umschalt Tab		Versetzt den Cursor in den vorherigen Fensterausschnitt.
Umschalt →		Nächsten Fensterausschnitt
Umschalt ←		gemäß der
Umschalt ↑		Pfeilrichtung
Umschalt ↓		aktivieren
Esc		Schließt ein *Inspector*-Fenster, Menü oder Dialogfenster.
Einfg		Anfang einer Blockauswahl (hervorgehoben).
Tab	*Window/Next pane*	Versetzt den Cursor in den nächsten Fensterausschnitt.

Die Befehle der Menüleiste

Die Menüleiste wird mit der Taste F10 aufgerufen. Die einzelnen Menüs können Sie dann aktivieren, indem Sie entweder

- den Auswahlbalken auf die Menübezeichnung positionieren und CR drücken;
- oder den ersten Buchstaben der Menübezeichnung eingeben.

Sie können das Menü mit Alt und dem ersten Buchstaben des gewünschten Menüs auch direkt öffnen.

Das System-Menü

Restore standard	Standard-Anordnung der Fenster wiederherstellen.
Repaint desktop	Gesamten Bildschirm neu aufbauen.
About	Informationen über Turbo Debugger.

Das File-Menü

Open	Neues Programm zum Debuggen laden.
Change dir	Verzeichnis oder Laufwerk wechseln.
Get info	Informationen über das Programm darstellen.
Symbol load	Symboltabelle getrennt laden.
Quit	Rückkehr zu Windows.

Das Edit-Menü

Copy	Kopiert einen Eintrag in die Zwischenablage.
Paste	Überträgt einen Eintrag aus der Zwischenablage in ein (Dialog-) Fenster.
Copy to log	Kopiert den selektierten Eintrag oder den Eintrag, bei dem sich der Cursor befindet, in das *Log*-Fenster.

Das View-Menü

Breakpoints	Breakpoints im Fenster darstellen.
Stack	Stack mit den Funktionsaufrufen darstellen.
Log	Protokoll von Ereignissen und Daten im Fenster darstellen.
Watches	Variablen zur Überwachung darstellen.
Variables	Globale und lokale Variablen darstellen.
Module	Quellcode darstellen.
File	Dateien im ASCII-Format oder hexadezimal darstellen.
CPU	CPU-Befehle, Daten und Stack darstellen.
Dump	Daten hexadezimal darstellen.
Registers	CPU-Register und Flags darstellen.
Numeric processor	Koprozessor oder Emulator darstellen.
Execution history	Ausgeführte Programmschritte und aufgezeichnete Tastatureingaben für Rückwärts-Ausführung darstellen.
Hierarchy	Objekt- oder Klassenliste und Hierarchiebaum darstellen.
Windows messages	Eine Liste von Windows-Meldungen für ein oder mehrere Fenster in Ihrem Anwendungsprogramm anzeigen.

Clipboard	Zeigt das *Clipboard* -Fenster mit dem Inhalt der Zwischenablage an.
Another	
Module	Zusätzliches *Module*-Fenster anlegen.
Dump	Zusätzliches *Dump*-Fenster anlegen.
File	Zusätzliches *File*-Fenster anlegen.

Das Run-Menü

Run	Programm starten.
Go to cursor	Programm läuft bis zur aktuellen Cursor-Position.
Trace into	Eine Quellzeile oder einen Befehl ausführen.
Step over	Einen Befehl oder eine Quellzeile ausführen, (Unterprogramme werden komplett ausgeführt).
Execute to	Programm bis zur angegebenen Adresse ausführen.
Until return	Programm bis zur Rückkehr aus der Funktion/Prozedur ausführen.
Animate	Programm Befehl für Befehl verlangsamt ausführen.
Back trace	Rückwärtige Ausführung eines Programmschritts.
Instruction trace	Einzelnen Befehl ausführen.
Arguments	Kommandozeilenargumente festlegen.
Program reset	Aktuelles Programm neu laden.

Das Breakpoints-Menü

Toggle	Breakpoint setzen/löschen.
At	Breakpoint an einer bestimmten Adresse setzen.
Changed memory global	Globalen Breakpoint für Veränderungen eines Speicherbereichs setzen.
Expression true global	Globalen Breakpoint für einen Ausdruck setzen.
Hardware Breakpoint	Hardware-Breakpoint setzen.
Delete all	Alle Breakpoints löschen.

Das Data-Menü

Inspect	Datenobjekt untersuchen.
Evaluate/Modify	Ausdruck auswerten.
Add Watch	Variable im *Watches* -Fenster eintragen.
Function return	Funktionsergebnis einer Routine untersuchen.

Das Options-Menü

Language	Syntax für Ausdrücke entsprechend dem Quellmodul wählen.
Macros	
Create	Tastaturmakro definieren.
Stop recording	Makroaufnahme beenden.
Remove	Makro löschen.
Delete all	Alle Makros löschen.
Display options	Bildschirmoptionen festlegen.
Path for source	Pfad für Quelldateien.
Save options	Optionen, Makros und Fenstereinstellung speichern.
Restore options	Optionen laden.

Das Window-Menü

Zoom	Fenster auf volle Bildschirmgröße oder auf Ausgangsgröße bringen.
Next	Aktiviert nächstes geöffnetes Fenster.
Next pane	Nächsten Fensterausschnitt aktivieren.
Size/Move	Position und Größe eines Fensters ändern.
Iconize/Restore	Fenster in Symbol verwandeln und umgekehrt.
Close	Fenster schließen.
Undo close	Zuletzt geschlossenes Fenster öffnen.
Dump pane to log	Aktuellen Fensterausschnitt in *Log*-Fenster schreiben.
User screen	Ausgabebildschirm darstellen.
257Numbered window list256	Liste der geöffnete Fenster anzeigen.
Window pick	Fenster aus der Liste offener Fenster auswählen.

Das Help-Menü

Index	Aufruf des Hilfe-Index.
Previous topic	Vorletzten Hilfe-Bildschirm.
Help on help	Hilfestellungen zum Hilfesystem.

Die Befehle der lokalen Menüs

Sie können das lokale Menü eines Fensters mit Alt F10 aufrufen. Wenn sie die Strg -Tastenkürzel (über das Installationsprogramm TDWINST) aktiviert haben, können sie die einzelnen Menübefehle auch mit einer Kombination aus Strg und dem ersten Buchstaben des Befehls aktivieren.

Jedes Fenster (*Breakpoints* , *CPU* usw.) und jeder Ausschnitt in einem Fenster hat ein eigenes lokales Menü. In den folgenden Abschnitten beschreiben wir das lokale Menü für jedes Fenster und jeden Fensterausschnitt.

Einige Fensterausschnitte verfügen über Tastenkürzel für häufig verwendete Befehle ihrer lokalen Menüs. Im folgenden Abschnitt werden diese Spezialtasten vor den Menübefehlen für den entsprechenden Ausschnitt aufgegeben. In vielen Ausschnitten wird die CR-Taste als Kurzform zum Untersuchen und Verändern des gerade hervorgehobenen Objekts verwendet. Die Entf -Taste dient häufig als Kurzform für einen Löschbefehl in einem lokalen Menü. In einigen Ausschnitten können Sie einfach Zahlen und Buchstaben eingeben, ohne daß Sie einen Befehl in einem lokalen Menü auswählen müssen. In diesen Fällen erscheint sofort ein Dialogfenster für das entsprechende Datenobjekt.

Das lokale Menü des Breakpoints-Fensters

Das *Breakpoints* -Fenster hat zwei Ausschnitte: links den Listenausschnitt und rechts den Detailausschnitt. Nur der Listenausschnitt weist ein lokales Menü auf.

Set options	Breakpoint-Aktionen, Bedingungen, Durchlaufzähler festlegen und ein-/ausschalten.
Add	Breakpoint einfügen.
Remove	Hervorgehobenen Breakpoint löschen.
Delete all	Alle Breakpoints löschen.
Inspect	Code untersuchen, auf dem der Breakpoint gesetzt ist.
Group	Gruppen von Breakpoints bearbeiten.

Das Tastenkürzel für *Remove* lautet in diesem Fenster Entf .

Die Menüs des CPU-Fensters

Das *CPU*-Fenster hat sechs Ausschnitte, die alle ein lokales Menü besitzen: Code-Ausschnitt, Selector-Ausschnitt, Datenausschnitt, Stack-Ausschnitt, Registerausschnitt und Statusausschnitt.

Das lokale Menü des Code-Ausschnitts

Goto	Code der neuen Adresse darstellen.

Origin	über CS:IP referenzierten Code darstellen.
Follow	Code der Adresse eines JMP- oder CALL-Ziels darstellen.
Caller	Code der aufrufenden Funktion darstellen.
Previous	Code der letzten Adresse darstellen.
Search	Befehle oder Bytes suchen.
View source	Umschalten auf *Module*-Fenster.
Mixed	Auswahl der gemischten Darstellung von Quelltext und disassemblierten Befehlen: *No/Yes/Both*
New CS:IP	CS:IP auf neue Adresse setzen.
Assemble	Befehl an der Cursor-Position assemblieren.
I/O	Menü zur Ein-/Ausgabe auf Port aufrufen.
In byte	Byte von Port einlesen.
Out byte	Byte an Port ausgeben.
Read word	Word von Port lesen.
Write word	Word an Port ausgeben.

Die Eingabe von Zeichen wird als Kurzform des *Assemble*-Befehls interpretiert.

Das lokale Menü des Selector-Ausschnitts

Selector	Auswahl eines neuen Selektors.
Examine	Inhalt des vom aktuellen Selektor adressierten Speicherbereichs im Daten- oder Code-Ausschnitt anzeigen.

Das lokale Menü des Datenausschnitts

Goto	Daten der neuen Adresse darstellen.
Search	Strings oder Byte-Folgen suchen.
Next	Suchbefehl wiederholen.
Change	Daten an der Cursor-Position verändern.
Follow	
Near code	Code-Ausschnitt auf ausgewählte NEAR-Adresse setzen.
Far code	Code-Ausschnitt auf ausgewählte FAR-Adresse setzen.
Offset to data	Datenausschnitt auf ausgewählte NEAR-Adresse setzen.
Segment: Offset to data	Datenausschnitt auf ausgewählte FAR-Adresse setzen.
Base segment:0 to Data	Datenausschnitt auf den Anfang des Segments setzen, das die durch die aktuelle Cursor-Position markierte Adresse erhält.
Previous	Daten der letzten Adresse darstellen.
Display as	
Byte	Daten als Hexadezimal-Bytes darstellen.
Word	Daten als Hexadezimal-Words darstellen.
Long	Daten als Hexadezimal-DWords darstellen.
Comp	Daten als 8-Byte-Comp-Integer darstellen (Pascal).
Float	Daten als 4-Byte-Fließkommazahl darstellen.
Real	Daten als 6-Byte-Fließkommazahl darstellen (Pascal).

Double	Daten als 8-Byte-Fließkommazahl darstellen.
Extended	Daten als 10-Byte-Fließkommazahl darstellen.
Block	
Clear	Block löschen.
Move	Block verschieben.
Set	Block initialisieren.
Read	Block von Datei in Speicher einlesen.
Write	Block von Speicher auf Datei schreiben.

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Statusausschnitts

Toggle	Hervorgehobenes Flag setzen oder löschen.

Die Eingabetaste (CR) oder die Leertaste wird als Kurzform des Befehls interpretiert.

Das lokale Menü des Registerausschnitts

Increment	Inhalt des hervorgehobenen Registers um 1 erhöhen.
Decrement	Inhalt des hervorgehobenen Registers um 1 erniedrigen.
Zero	Inhalt des hervorgehobenen Registers löschen.
Change	Inhalt des hervorgehobenen Registers ändern.
Registers 32-bit	Umschalten zwischen 16- und 32-Bit-Darstellung: *No/Yes*.

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Stack-Ausschnitts

Goto	Stack an der neuen Adresse darstellen.
Origin	Stack referenziert durch SS:SP darstellen.
Follow	Stack mit der Adresse des hervorgehobenen Objekts darstellen.
Previous	Letzte Adresse darstellen.
Change	Wert am Stack verändern.

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das Fenster Dump

Das Dump-Fenster entspricht dem Datenausschnitt des *CPU*-Fensters. Das lokale Menü ist identisch mit dem lokalen Menü des Datenausschnitts.

Die lokalen Menüs des Fensters Execution History

Das Fenster *Execution History* hat zwei Ausschnitte, von denen jeder ein lokales Menü beinhaltet: Den Ausschnitt *Instructions* (Befehle) und den Ausschnitt *Keystroke Recording* (Aufzeichnung der Tastatureingaben).

Das lokale Menü des Instructions-Ausschnitts

Im Ausschnitt *Instructions* werden bereits ausgeführte Befehle aufgelistet, damit Sie die Befehle anschauen oder rückgängig machen können.

Inspect	Bringt Sie zu dem hervorgehobenen Befehl.
Reverse Execute	Läßt das Programm bis zu dem hervorgehobenen Befehl rückwärts ablaufen.
Full History	Schaltet den Rückwärtslauf des Programm ein (*On*) oder aus (*Off*).

Das lokale Menü des Ausschnitts Keystroke Recording

Im Ausschnitt *Keystroke Recording* werden die aufgezeichneten Tastatureingaben angezeigt. Anhand dieser Aufzeichnung können Sie den Quellcode überprüfen oder das Programm erneut ablaufen lassen.

Inspect	Zeigt Ihnen die Zeile im Quellcode oder den Befehl, bei der/dem die Taste gedrückt wurde.
Keystroke Restore	Lädt und startet Ihr Programm im Kontext der hervorgehoben dargestellten Tastatureingabe (sehr nützlich wenn *Execution History* nicht verfügbar ist).

Das File-Fenster

Das *File*-Fenster stellt einen Dateiinhalt hexadezimal oder als ASCII-Text dar. Das lokale Menü dieses Fensters verfügt über folgende Befehle:

Goto	Datei ab Zeilennummer oder Hex-Offset darstellen.
Search	String oder Byte-Folge suchen.
Next	Suche wiederholen.
Display as	ASCII/Hex: Darstellungsmodus umschalten.
File	Neue Datei darstellen.

Die Eingabe von Zeichen wird als Kurzform des *Search*-Befehls interpretiert.

Das lokale Menü des Log-Fensters

Das *Log*-Fenster zeigt die Meldungen, die ins Protokoll geschrieben werden.

Open log file	Meldungen im Log-Fenster in Datei schreiben.

Close log file	Log-Datei schließen.
Logging	Protokoll ein- oder ausschalten: *No/Yes*.
Add comment	Kommentar in *Log*-Fenster einfügen.
Erase log	Meldungen im *Log*-Fenster löschen.
Display	Meldungen im Windows-Fenster anschauen, zur Auswahl stehen lokaler/globaler Heap oder Modulliste.

Die Eingabe von Zeichen wird als Kurzform des Befehls *Add comment* interpretiert.

Das Module-Fenster

Das *Module*-Fenster zeigt die Quelldatei des Programmmoduls.

Inspect	Inhalt der Variablen an Cursor-Position untersuchen.
Watch	Variable an Cursor-Position in *Watches* -Fenster aufnehmen.
Module	In ein anderes Modul wechseln.
File	Andere Datei darstellen.
Previous	Letztes Modul an der entsprechenden Position darstellen.
Line	Zeilennummer im Modul darstellen.
Search	String suchen.
Next	Suche wiederholen.
Origin	Aktuelle Programmposition darstellen.
Goto	Modul ab neuer Position darstellen.

Die Eingabe von Zeichen wird als Kurzform des *Goto*-Befehls interpretiert.

Die lokalen Menüs des Fensters Windows Messages

Das Fenster *Windows Messages* hat drei Aussschnitte: den Ausschnitt *Window Selection*, den Ausschnitt *Message Class* und den Ausschnitt *Messages*.

Das lokale Menü des Ausschnitts Windows Selection

Add	Fügt einen Fensternamen oder Handle-Wert hinzu.
Remove	Löscht das selektierte Fenster.
Delete all	Löscht alle Fensterauswahlen.

Die Eingabe von Zeichen wird als Kurzform des *Add*-Befehls interpretiert.
Die Entf -Taste und die Tastenkombination Strg Y sind Tastenkürzel für den Befehl *Remove*.

Das lokale Menü des Ausschnitts Message Class

Add	Fügt eine Meldungsklasse oder eine einzelne Meldung hinzu.
Remove	Löscht die selektierte Meldungsklasse oder Einzelmeldung.
Delete all	Löscht alle Meldungsklassen- oder Einzelmeldungsauswahlen.

Die Eingabe von Zeichen wird als Kurzform des *Add*-Befehls interpretiert.
Die Entf -Taste und die Tastenkombination Strg Y sind Tastenkürzel für den Befehl *Remove*.

Das lokale Menü des Ausschnitts Messages

Send log to window	Sendet alle empfangenen Meldungen an das *Log*-Fenster, so daß sie in einer Log-Datei gespeichert werden können.
Erase log	Löscht alle Meldungen in diesem Ausschnitt.

Das Clipboard-Fenster

Das Clipboard-Fenster zeigt alle Einträge, die in die Zwischenablage kopiert wurden. Es hat einen Ausschnitt dessen lokales Menü folgende Befehle enthält:

Inspect	Wechselt in das Fenster, aus dem der Eintrag kopiert wurde, damit Sie den Eintrag untersuchen können.
Remove	Löscht den aktuellen Eintrag.
Delete All	Löscht alle Einträge der Zwischenablage.
Freeze	Fixiert aktuellen Wert des hinterlegten Eintrags.

Die lokalen Menüs des Fensters Numeric processor

Das Fenster *Numeric Processor* hat drei Ausschnitte: Register, Status und Control

Das lokale Menü des Registerausschnitts
Die Tasten Z, E und C sind Kurzformen der lokalen Menübefehle in diesem Ausschnitt.

Zero	Löscht das hervorgehobene Register.
Empty	Setzt das hervorgehobene Register auf leer.
Change	Verändert den Inhalt des hervorgehobenen Registers.

Die Eingabe von Zeichen wird als Kurzform des *Change*-Befehls interpretiert.

Das lokale Menü des Statusausschnitts

Toggle	Flag-Werte durchschalten.

Die Eingabetaste CR oder die Leertaste Leertaste ist eine Kurzform des lokalen Menübefehls in diesem Ausschnitt.

Das lokale Menü des Control-Ausschnitts

Toggle	Flag-Werte durchschalten.

Die Eingabetaste CR oder die Leertaste Leertaste ist eine Kurzform des lokalen Menübefehls in diesem Ausschnitt.

Das Hierarchy-Fenster

Das *Hierarchy*-Fenster hat zwei Ausschnitte: eine Liste mit den Objekttypen/Klassen und einen Hierarchiebaum. Außerdem kann ein dritter Ausschnitt, der Vorfahrenbaum, vorhanden sein, wenn ein Programm in C++ die Mehrfachvererbung ausnutzt.

Das lokale Menü des Class Auschnitts

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen.
Tree	Wechsel in den Hierarchiebaum.

Das lokale Menü des Hierachy-Tree-Auschnitts

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen.
Parents	Wechsel in den Vorfahrenbaum, falls unter C++ die Mehrfachverbung verwendet wird.

Das lokale Menü des Parent-Tree-Auschnitts

Inspect	Inhalt des ausgewählten Objekt- oder Klassentyps anzeigen.

Das Registers-Fenster

Das Fenster *Registers* entspricht den Register- und Statusausschnitten des *CPU*-Fensters. Die lokalen Menüs sind identisch zum lokalen Menü des Registerausschnitts und Statusausschnitts.

Das lokale Menü des Stack-Fensters

Das *Stack*-Fenster zeigt die gerade aktivierten Funktionen.

Inspect	Quellcode der hervorgehobenen Funktion darstellen.
Locals	Argumente der Funktion anzeigen.

Die Eingabetaste CR ist eine Kurzform zum lokalen Menübefehl *Inspect*.

Das Variables-Fensters

Das *Variables* -Fenster enthält zwei Ausschnitte, die beide über ein lokales Menü verfügen: den Auschnitt Global Symbol und den Ausschnitt Local Symbol

Das lokale Menü des Global-Symbol-Auschnitts

Inspect	Zeigt den Inhalt des hervorgehobenen Symbols.
Change	Verändert den Inhalt des hervorgehobenen Symbols.
Watches	Fügt das hervorgehobene Symbol im Fenster *Watches* hinzu.

Die Eingabetaste CR ist eine Kurzform zum lokalen Menübefehl *Inspect*.

Das lokale Menü des Local-Symbol-Ausschnitts

Inspect	Zeigt den Inhalt des hervorgehobenen Symbols.
Change	Verändert den Inhalt des hervorgehobenen Symbols.
Watches	Fügt das hervorgehobene Symbol im Fenster *Watches* hinzu.
Show	Öffnet das Dialogfenster Show, welches Ihnen folgende Optionen bietet:
Static	Nur statische Variablen anzeigen.
Auto	Nur lokale Variablen des aktuellen Blocks anzeigen.
Both	Beide Arten von Variablen anzeigen (Standardeinstellung).
Module	Aktuelles Modul wechseln.

Die Eingabetaste CR ist eine Kurzform des lokalen Menübefehls *Inspect*.

Das Fenster Watches

Das *Watches* -Fenster hat einen einzigen Ausschnitt, der die Namen und die Werte der Variablen zeigt, die Sie beobachten. Das lokale Menü des Watches-Fensters verfügt über folgende Befehle:

Watch	Variable einfügen.
Edit	Variablennamen oder Ausdruck editieren.
Remove	Hervorgehobene Variable löschen.
Delete all	Alle *Watches* -Variablen löschen.
Inspect	Inhalt der hervorgehobenen Variablen untersuchen.
Change	Inhalt der hervorgehobenen Variablen verändern.

Die folgenden Tasten sind Kurzformen der lokalen Menübefehle in diesem Fenster.

Taste	Funktion
beliebiges Zeichen	*Watch*
CR	*Edit*
Entf	*Remove*

Das lokale Menü des Inspector-Fensters

Ein *Inspector*-Fenster zeigt den Inhalt eines Datenobjekts.

Range	Wählt Array-Elemente für die Untersuchung aus.
Change	Verändert den Wert des hervorgehobenen Objekts.
Inspect	Öffnet ein *Inspector*-Fenster für das selektierte Datenobjekt.
Descend	Erweitert das hervorgehobene Datenobjekt in diesem *Inspector*-Fenster.
New expression	Untersucht einen neuen Ausdruck in diesem *Inspector*-Fenster.
Type cast	Datenvorgabe für das ausgewählte Datenobjekt.

Das Inspector-Fenster für Objekttypen/Klassen

Das *Inspector*-Fenster für Objekttypen/Klassen hat zwei Ausschnitte, die den Inhalt (Datenfelder oder -elemente und Methoden oder Elementfunktionen) einer Klasse oder eines Objekts anzeigen. Die lokalen Menüs, die in beiden Ausschnitten gleich sind, unterscheiden sich sehr stark von den üblichen *Inspector*-Fenstern.

Inspect	*Inspector*-Fenster für den ausgewählten Typ öffnen.
Hierarchy	Rückkehr zum *Hierarchy*-Fenster.
Show inherited	Umschalten zwischen der Darstellung des Inhalts des Objekts/Klasse inklusive aller vererbten Elemente und dem Inhalt des aktuellen Objekts oder Klasse.

Das Inspector-Fenster für Objekt-/Klasseninstanzen

Das *Inspector*-Fenster für Instanzen von Objeten oder Klassen enthält drei Ausschnitte, von denen nur die ersten beiden ein lokales Menü besitzen. (Der dritte gibt nur den Objekttyp oder die Klasse an, zu der die Instanz gehört). Die beiden lokalen Menüs sind abgesehen vom *Change*-Befehl gleich und enthalten folgenden Befehle:

Range	Array-Elemente für die Untersuchung auswählen.
Change	Wert des ausgewählten Datenojekts ändern.
Methods	Umschalten, ob Methoden oder Elementfunktionen im mittleren Ausschnitt zusammengefaßt werden.
Show inherited	Umschalten zwischen der Darstellung des Inhalts des Objekts/Klasse inklusive aller vererbten Elemente und dem Inhalt des aktuellen Objekts oder Klasse.
Inspect	*Inspector*-Fenster für das aktuelle Element öffnen.

Descend	Ausgewähltes Datenobjekt in einem *Inspector*-Fenster erweitern.
New expression	Untersucht einen neuen Ausdruck in diesem *Inspector*-Fenster.
Type cast	Datenvorgabe für das ausgewählte Datenobjekt.
Hierarchy	Rückkehr zum *Hierarchy*-Fenster.

Die Textausschnitte

Die Bezeichnung Textausschnitt ist eine allgemeine Bezeichnung für alle Ausschnitte, in denen der Inhalt einer Textdatei dargestellt wird. Der blinkende Cursor gibt die aktuelle Position in der Datei an. In der folgenden Tabelle finden Sie alle in solchen Ausschnitten verfügbaren Befehle:

Tabelle 14.2 Tastaturbefehle in Textausschnitten

Taste	Funktion
Einfg	Markiert einen Textblock.
↑	Bewegt den Cursor um eine Zeile nach oben.
↓	Bewegt den Cursor um eine Zeile nach unten.
→	Bewegt den Cursor um eine Spalte nach rechts.
←	Bewegt den Cursor um eine Spalte nach links.
Strg →	Bewegt den Cursor zum nächsten Wort.
Strg ←	Bewegt den Cursor zum vorherigen Wort.
Pos1	Bewegt den Cursor an den Anfang der Zeile.
Ende	Bewegt den Cursor an das Ende der Zeile.
Bild↑	Blättert eine Bildschirmseite nach oben.
Bild↓	Blättert eine Bildschirmseite nach unten.
Strg Pos1	Bewegt den Cursor in die erste Zeile des Ausschnitts.
Strg Ende	Bewegt den Cursor in die letzte Zeile des Ausschnitts.
Strg Bild↑	Bewegt den Cursor in die erste Zeile der Datei.
Strg Bild↓	Bewegt den Cursor in die letzte Zeile der Datei.

Wenn Sie die Strg -Tastenkürzel nicht verwenden, dann können Sie in einem Textausschnitt auch die Wordstar-Befehle verwenden.

Die Listenausschnitte

Die Bezeichnung Listenausschnitt ist eine allgemeine Bezeichnung für alle Ausschnitte, in denen Listen von Datenobjekten dargestellt werden. Der Auswahlbalken gibt die aktuelle Position in der Liste an. In der folgenden Tabelle finden Sie alle in solchen Ausschnitten verfügbaren Befehle:

Tabelle 14.3 Tastaturbefehle in Listenausschnitten

Taste	Funktion
↑	Bewegt den Auswahlbalken um eine Zeile nach oben.
↓	Bewegt den Auswahlbalken um eine Zeile nach unten.
→	Bildlauf nach rechts.
←	Bildlauf nach links.
Pos1	Bewegt den Auswahlbalken an den Anfang der Zeile.
Ende	Bewegt den Auswahlbalken an das Ende der Zeile
Bild↑	Blättert eine Bildschirmseite nach oben.
Bild↓	Blättert eine Bildschirmseite nach unten.
Strg Pos1	Bewegt den Auswahlbalken in die erste Zeile des Ausschnitts.
Strg Ende	Bewegt den Auswahlbalken in die letzte Zeile des Ausschnitts.
Strg Bild↑	Bewegt den Auswahlbalken zum ersten Eintrag in der Liste.
Strg Bild↓	Bewegt den Auswahlbalken zum letzten Eintrag in der Liste.
Rücktaste	Nimmt ein Zeichen bei der schrittweisen Suche zurück.
Buchstabe	Schrittweises Suchen (Auswahl durch Eintippen).
Einfg	Markiert mehrere Einträge für Blockkopie.

Wenn Sie die Strg -Tastenkürzel nicht verwenden, dann können Sie in einem Listenausschnitt auch die Wordstar-Befehle verwenden.

Funktionstasten in Eingabefeldern und Eingabeaufzeichchnungsfeldern

Die folgende Tabelle zeigt, welche Funktionstasten innerhalb eines Eingabefeldes zur Verfügung stehen.

Tabelle 14.4 Tastaturbefehle in Eingabefeldern

Taste	Funktion
↑	Bewegt den Cursor um einen Eintrag nach oben.
↓	Bewegt den Cursor um einen Eintrag nach unten.

Taste	Funktion
→	Bewegt den Cursor um ein Zeichen nach rechts.
←	Bewegt den Cursor um ein Zeichen nach links.
Strg →	Bewegt den Cursor zum nächsten Wort
Strg ←	Bewegt den Cursor zum vorherigen Wort.
Pos1	Bewegt den Cursor an den Anfang der Zeile.
Ende	Bewegt den Cursor an das Ende der Zeile.
Bild↑	Blättert eine Bildschirmseite nach oben.
Bild↓	Blättert eine Bildschirmseite nach unten.
Strg Pos1	Bewegt den Cursor in die erste Zeile des Ausschnitts.
Strg Ende	Bewegt den Cursor in die letzte Zeile des Ausschnitts.
Strg Bild↑	Bewegt den Cursor in die erste Zeile der Liste.
Strg Bild↓	Bewegt den Cursor in die letzte Zeile der Liste.
Rücktaste	Löscht das Zeichen links vom Cursor.
CR	Akzeptiert die Eingabe.
Entf	Löscht das Zeichen an der Cursor-Position.
Esc	Bricht die Eingabe ab und kehrt zum Menü zurück.
Strg N	Vervollständigt einen teilweise eingegebenen Namen in einem Eingabefeld.

Befehle zum Verändern der Fensteranordnung

Die folgende Tabelle zeigt die Tasten, mit den Sie ein Fenster verschieben oder seine Größe verändern können.

Tabelle 14.5 Tastaturbefehle zum Verändern der Fensteranordnung

Taste	Funktion
Strg F5	Schaltet die Fensterpositionierung um.
↑	Bewegt das Fenster um eine Zeile nach oben.
↓	Bewegt das Fenster um eine Zeile nach unten.
→	Bewegt das Fenster um eine Spalte nach rechts.
←	Bewegt das Fenster um eine Spalte nach links.
Umschalt ↑	Verkleinert das Fenster von unten her.
Umschalt ↓	Vergrößert das Fenster nach unten.
Umschalt ←	Verkleinert das Fenster von rechts her.
Umschalt →	Vergrößert das Fenster nach rechts.
Pos1	Bewegt das Fenster auf die linke Seite des Bildschirms.
Ende	Bewegt das Fenster auf die rechte Seite des Bildschirms.

Taste	Funktion
Bild↑	Bewegt das Fenster in die oberste Zeile des Bildschirms.
Bild↓	Bewegt das Fenster in die unterste Zeile des Bildschirms.
CR	Akzeptiert die aktuelle Position.
Esc	Bricht die Fensterpositionierung ab.

Die Jokerzeichen

Sie können die Jokerzeichen in zwei Fällen verwenden:

- wenn Sie einen Dateinamen eingeben, um eine Datei zu laden oder zu untersuchen;
- wenn Sie einen String in einem Textausschnitt suchen.

Das Fragezeichen (?) steht für ein beliebiges Zeichen im Such-String. Das Sternchen (*) wird stellvertretend für 0 oder mehrere Zeichen im Such-String angegeben.

Der vollständige Menübaum

In Abbildung 14.1, auf den folgenden Seite, ist die komplette Struktur der Turbo-Debugger-Menüs dargestellt.

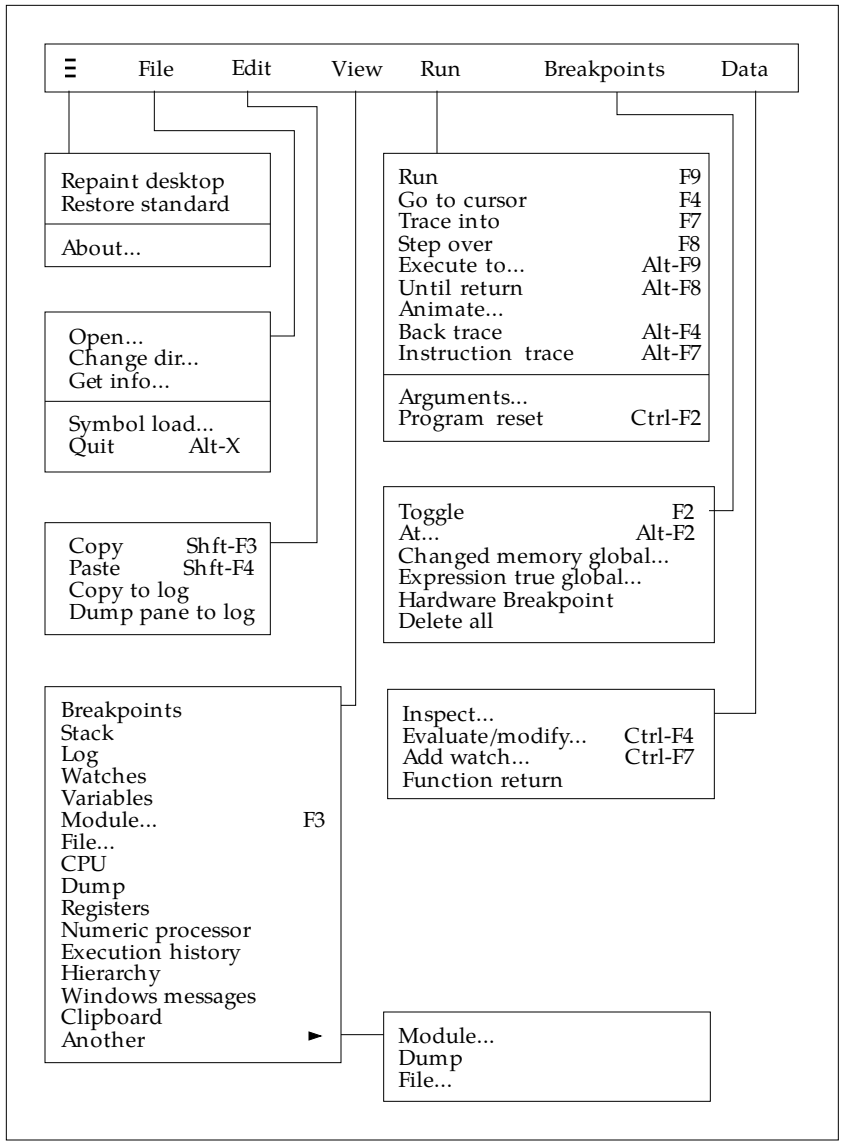


Abbildung 14.1 Der Menübaum von Turbo Debugger

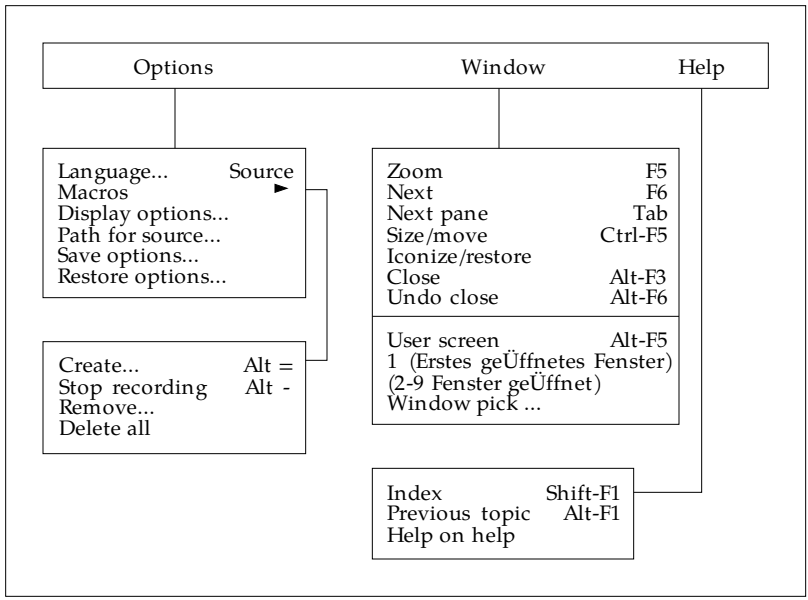


Abbildung 14.1 (Fortsetzung) Der Menübaum von Turbo Debugger

15. Ein Standard C-Programm debuggen

Das Debuggen von Programmen ist, wie die anderen Phasen der Programmerstellung, zum Teil eine Wissenschaft, zum Teil aber auch eine Kunst. Es gibt bestimmte Regeln, um einen Fehler aufzuspüren, während man in der gleichen Zeit mit etwas Intuition einen umständlichen Weg verkürzen kann.

Je mehr Programme Sie debuggen, desto schneller werden Sie die Ursachen für Probleme in Ihrem Programm finden. Sie werden Methoden kennenlernen, die Ihnen dabei entgegenkommen, und Sie werden lernen, wie man Programmiermethoden ändert, die Ihnen immer wieder Probleme bereiten.

In diesem Kapitel besprechen wir einige unterschiedliche Wege beim Debuggen, unterhalten uns über die verschiedenen Arten von Fehlern, die vielleicht in Ihren Programmen auftauchen können und schlagen einige Methoden vor, wie Sie Ihr Programm testen können, damit es zuverlässig läuft und auch in Zukunft laufen wird.

Wir wollen einmal sehen, wo anzufangen ist, wenn Sie ein Programm haben, das nicht richtig arbeitet.

Wenn's nicht läuft

Zuallererst: Keine Panik! Auch der erfahrenste Programmierer schreibt selten auf Anhieb ein fehlerfreies Programm. Um nicht allzu viel Zeit zu verlieren, muß man der Versuchung widerstehen können, wahllos herumzuraten, wo der Fehler liegen könnte.

Gehen Sie von einer Reihe von Annahmen aus, und testen Sie jede einzeln durch. Sie könnten zum Beispiel folgende Hypothese aufstellen: »Der Fehler muß vor dem Aufruf der Funktion *xyz* auftreten.« Anschließend prüfen Sie diese Hypothese, indem Sie Ihr Programm direkt vor dem Funktionsaufruf von *xyz* anhalten und nachschauen, ob sich bis dahin ein Problem ergeben hat. Wenn Sie an diesem Punkt ein Problem entdecken, können Sie in einer neue Hypothese annehmen, daß das Problem noch viel früher in Ihrem Programm auftritt.

Wenn aber bis zum Funktionsaufruf von *xyz* alles in Ordnung ist, dann war Ihr erster Ausgangspunkt falsch. Sie müssen nun Ihre Hypothese wie folgt verändern: »Der Fehler tritt nach dem Funktionsaufruf von *xyz* auf.« Indem Sie

eine Reihe von Tests wie diesen ausführen, werden Sie sehr bald den Programmbereich finden, in dem das Problem verursacht wird.

Das ist alles schön und gut, werden Sie vielleicht jetzt sagen. Wie aber kann ich feststellen, daß das Programm korrekt arbeitet, wenn ich das Programm anhalte? Eine der besten Methoden, das Verhalten eines Programms zu überprüfen, ist die Untersuchung der Variablen und Datenobjekte im Programm. Wenn Sie eine Routine haben, die den Inhalt eines Array löscht, dann können Sie die Operation überprüfen, indem Sie das Programm nach der Ausführung der Funktion anhalten und dann überprüfen, ob jedes Array-Element auch tatsächlich gelöscht wurde.

Der persönliche Stil beim Debuggen

Jeder hat seinen eigenen Programmierstil und entwickelt auch beim Debuggen seinen eigenen Stil. Die Vorschläge zum Debuggen, die wir Ihnen hier machen, sind lediglich ein Ausgangspunkt, von dem aus Sie Ihre eigenen Methoden entwickeln können.

Oft bestimmt der Verwendungszweck, wie Sie mit dem Debuggen anfangen. Einige Programme werden nur von Ihnen selbst eingesetzt oder nur ein- oder zweimal für eine bestimmte Aufgabe verwendet. Bei diesen Programmen ist ein vollständiger Test aller Komponenten wahrscheinlich Zeitverschwendung. Dies ist insbesondere dann der Fall, wenn Sie die Richtigkeit des Programms anhand der Ausgaben überprüfen können. Ein Programm, das an andere Leute verteilt werden soll, oder dessen Richtigkeit sehr schwer nachzuvollziehen ist, werden Sie sehr viel intensiver austesten.

Einfach mal laufen lassen

Bei einfachen Programmen oder Gelegenheitsprogrammen ist es meistens am besten, wenn Sie es einfach starten und sehen, was passiert. Wenn Ihr Testfall zu Problemen führt, können Sie das Programm mit den einfachsten Eingaben starten und dann die Ausgabe wieder überprüfen. Sie können dann weitergehen und kompliziertere Eingaben überprüfen, bis die Ausgabe falsch wird. Damit haben Sie sofort einen guten Überblick, wieviel oder wie wenig in Ihrem Programm falsch ist.

Schrittweise testen

Wenn Sie ganz sicher sein wollen, daß Ihr Programm korrekt ist, müssen Sie sowohl die einzelnen Routinen überprüfen als auch einige Programmläufe mit Testdatensätzen durchführen. Sie haben dabei viele Möglichkeiten: Sie können

jede Routine sofort beim Schreiben überprüfen, indem Sie ein Testprogramm für diese Routinen erstellen. Man kann aber auch den Debugger beim fertigen Programm verwenden, um jede einzelne Routine durchzugehen.

Mögliche Fehler

Die Fehler in Ihren Programmen lassen sich in zwei große Kategorien einteilen: diejenigen, die typisch für die verwendete Sprache sind und diejenigen, die in allen Programmiersprachen und Programmierumgebungen auftreten.

Wenn Sie beim Debuggen Ihrer Programme bewußt vorgehen, werden Sie sowohl die sprachspezifischen Konstrukte, mit denen Sie Probleme haben, als auch Ihre allgemeinen Programmierfehler erkennen. Mit diesem Wissen können Sie versuchen, diese Fehler in Zukunft zu vermeiden. Sie haben außerdem einen guten Ausgangspunkt zur Fehlersuche in zukünftigen Programmen.

Es ist wichtig, daß Sie jeden Fehler als einen Spezialfall einer allgemeinen Sorte von Fehlern verstehen, damit Sie sich auf diese Weise bei der Erstellung fehlerfreier Programme verbessern können. Es ist allemal besser, wenn Sie fehlerfreien Code schreiben, als wenn Sie zu einem Experten in der Fehlersuche werden.

Allgemeine Fehler

Die folgenden Beispiele decken nur sehr oberflächlich die Vielzahl verschiedener Fehlermöglichkeiten in einem Programm ab.

Versteckte Seiteneffekte

Wenn Sie sorglos globale Variablen in Funktionen verwenden, kann ein Funktionsaufruf manchmal seltsame Ergebnisse in Variablen oder Datenstrukturen hinterlassen:

```
char   Arbeitspuffer[20] ;
strcpy ( Arbeitspuffer, "Alles fertig\n" ) ;
umwandeln ( "xyz" ) ;
printf ( Arbeitspuffer ) ;
...
umwandeln ( char *p )
    {
    strcpy ( Arbeitspuffer, p )
    while ( *p )
    ...
   }
```

In diesem Fall wäre es richtig gewesen, wenn sich die Funktion ihren eigenen *Arbeitspuffer* angelegt hätte.

Von initialisierten Daten darf man nicht ausgehen

Manchmal geht man davon aus, daß eine andere Routine bereits bestimmte Werte belegt hat:

```
char *ArbeitsPuffer ;
StringDrauf ( char *s )
    {
    strcpy ( ArbeitsPuffer, s ) ;   /* Hoppsalala */
    }
```

Man könnte diesen Code mit folgender Anweisung absichern:

```
if ( ArbeitsPuffer == NULL )
    ArbeitsPuffer = (char *) malloc ( 20 ) ;
```

Aufräumen vergessen

Diese Sorte von Fehler kann zu einem Programmabsturz führen, weil die Kapazität des Heap erschöpft ist:

```
string_verkleinern ( char *p )
    {
    char *arb = (char *) malloc (strlen (p)) ;
    strcpy ( arb, p ) ;
    ...
    return (p) ;    /* Hey! der Arbeitsbereich */
                    /* ist immer noch reserviert */
    }
```

Kleinigkeit mit großer Wirkung

Bei dieser Art von Fehlern wird eine Konstante um den Wert 1 zuviel oder zuwenig angegeben:

```
for ( n = 1 ; n < 10 ; n++ )
    {
    ...             /* Schleife wird nur 9 mal durchlaufen */
    }
```

In diesem Fall sieht man die Zahlen 1 und 10 und meint, daß die Schleife von eins bis zehn durchlaufen wird. Damit das Programmstück läuft, muß man jedoch den Operator < durch <= austauschen.

Typische Fehler in C

Turbo C++ für Windows ist sehr gut im Auffinden von C-spezifischen Fehlern, vor denen Sie von anderen C-Compilern überhaupt nicht gewarnt werden. Sie können viel Zeit sparen, wenn Sie den Compiler so konfigurieren, daß alle Warnungen, die der Compiler erzeugen kann, ausgegeben werden. (Sehen Sie dazu im Benutzerhandbuch von Turbo C++ für Windows nach.)

Die folgenden Ausführungen sind keineswegs vollständig für die Vielzahl an

Möglichkeiten, die in C Ärger machen können. Bei einigen dieser Fehler erhalten Sie von Turbo C++ für Windows eine Warnung. Merken Sie sich, daß Sie jeder Warnung auf den Grund gehen sollten, weil hinter jeder Warnung ein Fehler stecken kann.

Automatische Variablen wurden nicht initialisiert

In C hat jede als automatisch definierte Variable in einer Funktion solange keinen definierten Wert, bis Sie der Variablen einen Wert zuweisen:

```
zehnmal_wiederholen ()
    {
    int    n ;
    while ( n < 10 )
        {
        ...
        n++
        }
    }
```

In dieser Funktion ist nicht vorhersehbar, wie oft die *while*-Schleife ausgeführt wird, da *n* nicht mit 0 initialisiert war, bevor die Variable als Zähler verwendet wurde.

Verwechslung von = und ==

In C kann man sowohl eine Zuweisung als auch einen Gleichheitsoperator in einem Ausdruck verwenden. Zum Beispiel:

```
if ( x = y )
    {
    ...
    }
```

Damit wird der Inhalt von *y* versehentlich *x* zugewiesen. Die Anweisungen des if-Blocks werden, je nachdem, ob der Wert von *y* Null ist oder nicht, ausgeführt. Mit Sicherheit wollten Sie folgenden Ausdruck angeben:

```
if ( x == y )
    {
    ...
    }
```

Rangfolge der Operatoren vertauscht

In C gibt es so viele Operatoren, daß man leicht den Überblick über die Reihenfolge der Auswertung verlieren kann, wenn viele verschiedene Operatoren in einem Ausdruck verwendet werden. Einige der häufigsten Kombinationen, die leicht durcheinander gebracht werden, sind Schiebe-, Additions- oder Subtraktionsbefehle. Zum Beispiel ergibt:

```
x = 3 << 1 + 1 ;
```

den Wert 12 und nicht 7, wie Sie vielleicht erwartet haben, weil Sie davon ausgegangen sind, daß << vor + ausgeführt wird.

Falsche Zeigerarithmetik

Wenn Sie Zeiger für die Adressierung von Arrays verwenden, müssen Sie bei Additionen und Subtraktionen aufpassen. Im folgenden Beispiel

```
int *intp ;
intp += sizeof (int) ;
```

wird intp nicht etwa genau einmal inkremeniert, um das nächste Element des Array zu adressieren. *intp* wird nämlich um zwei Array-Elemente erhöht. Wenn Sie etwas zu einem Zeiger hinzuzählen oder etwas abziehen, berücksichtigt C die Größe des Datenobjekts, auf das der Zeiger verweist, so daß der Zeiger mit

```
intp++
```

schon das nächste Element adressiert.

Vorzeichenerweiterung übersehen

Sie müssen aufpassen, wenn Sie Integer-Zahlen verschiedener Größe einander zuweisen:

```
int     i = 0xFFFE ;
long    l ;

l = i ;
if ( l & 0x80000000 )
  {
  ...    /* Diese Anweisungen werden IMMER ausgeführt */
  }
```

Eine der Stärken von C kann Ärger machen, wenn man sich über die Folgen nicht im Klaren ist. In C können Sie skalare Werte beliebig zuweisen (*char, int,* usw.). Wenn Sie einen skalaren Integer einem größeren Integer-Typ zuweisen, bleibt das Vorzeichen (positiv oder negativ) in der größeren Integer-Variable erhalten, wobei das Vorzeichen-Bit in den oberen Bereich des größeren Integer-Werts erweitert wird. Aus einem *int*-Wert -2 (0xFFFE) wird zum Beispiel der *long*-Wert von -2 (0xFFFFFFFE).

Verlust signifikanter Bits

Dieses Beispiel ist das genaue Gegenteil des vorherigen:

```
int     i ;
long    l = 0x10000 ;

i = l ;
while ( i > 0 )
    {
    ...    /* wird NIE ausgeführt */
    }
```

In diesem Fall werden bei der Zuweisung von *l* nach *i* die oberen 16 Bits von *l* abgeschnitten, so daß in *i* der Wert 0 steht.

Überflüssige Strichpunkte

Das folgende Codefragment macht auf den ersten Blick einen guten Eindruck:

```
for ( x = 0 ; x < 10 ; x++ ) ;
    {
    ...       /* wird nur einmal ausgeführt */
    }
```

Warum wird der Code in den geschweiften Klammern nur ein einziges Mal ausgeführt? Mit einem gezielten Blick entdeckt man einen Strichpunkt (*;*) am Ende der *for*-Anweisung. Bei diesem schwierig zu findenden Fehler wird die Schleife zwar zehnmal ausgeführt, aber es wird nichts getan. Der nachfolgende Block wird nur einmal ausgeführt. Das ist eine ziemlich unangenehme Angelegenheit, denn mit den üblichen Methoden, die Formatierung und Einrückung der Blöcke zu überprüfen, findet man diesen Fehler nicht.

Makros mit Seiteneffekten

Das folgende Problem sollte eigentlich reichen, ein ganzes Leben lang auf die Verwendung von #*define*-Makros zu verzichten:

```
#define toupper(c) 'a'<=(c) && (c)<='z' ? (c)-'a'+'A' : (c)
char c, *p ;
c = toupper (*p++) ;
```

In diesem Beispiel wird *p* zwei- oder dreimal inkrementiert, je nach dem, ob das Zeichen ein Großbuchstabe ist. Diese Probleme sind sehr schwer zu finden, da die Seiteneffekte in der Makrodefinition versteckt sind.

Wiederholungen von automatischen Variablen

Ebenfalls sehr schwer zu finden ist der folgende Fehler:

```
myfunc ()
    {
    int    n ;
    for ( n = 5 ; n >= 0 ; n -- )
       {
       int n = 10 ;
       ...
       if ( n == 0 )
           {
           ...    /* wird niemals ausgeführt */
           }
       }
   }
```

In diesem Fall wird die automatische Variable *n* in einem inneren Block erneut verwendet, wodurch der Zugriff auf die im äußeren Block definierte

Variable unmöglich gemacht wird. Sie müssen sehr vorsichtig sein, wenn Sie Variablennamen in dieser Art und Weise mehrfach verwenden wollen. Diese Art von Ärger hat man schneller als man glaubt, da die meisten Programmierer nur eine begrenzte Anzahl von Variablennamen für Schleifenzähler verwenden (wie zum Beispiel *i, n* usw.).

Unzulässige Verwendung von automatischen Variablen
Folgende Funktion soll einen Zeiger auf das Ergebnis liefern:

```
int *Durch3Teilen ( int n )
    {
    int    i ;
    i = n / 3 ;
    return (&i) ;
    }
```

Das Problem besteht hier darin, daß die als automatisch gespeicherte Variable zum Zeitpunkt des Rücksprungs von der Funktion nicht mehr definiert ist und wahrscheinlich von anderen Stack-Parametern überschrieben wurde.

Funktionsergebnis nicht definiert
Wenn eine Funktion nicht mit einer *return*-Anweisung abgeschlossen wird, in der ein Ausdruck angegeben ist, wird ein unbestimmter Wert zurückgegeben. Zum Beispiel:

```
char *erster_grossbuchstabe ( char *p )
   {
   while ( *p )
       {
       if ( 'A' <= *p && *p <= 'Z' )
           return (p) ;
       p++ ;
       }
           /* hoppla - hier wird nichts zurückgegeben */
   }
```

Wenn im String keine Großbuchstaben vorkommen, dann ist das zurückgelieferte Ergebnis zufällig. Sie sollten zumindest eine Anweisung *return* (*0*) am Ende der Funktion einfügen.

Falsche Verwendung von break
Das reservierte Wort *break* verläßt nur eine Ebene einer *do-*, *for-*, *switch-* oder *while*-Schleife:

```
for (...)
   {
   while (...)
       {
       if (...)
           break;  /* es soll zwar die for-Schleife ver-*/
```

```
        }           /* lassen werden, aber man springt */
    }               /* nur aus der while-Schleife */
```

In diesem Fall wird mit der *break*-Anweisung nur die *while*-Schleife verlassen. Dies ist eine der wenigen Situationen, in denen die Verwendung einer *goto*-Anweisung einigermaßen gerechtfertigt ist.

Code hat keine Auswirkungen

Manchmal erhält man auch mit Tippfehlern einen ausgezeichnet compilierbaren Quellcode, der aber nicht das macht, was man will:

```
a + b ;
```

In diesem Fall war vielleicht $a \mathrel{+}= b$ beabsichtigt.

Richtigkeit überprüfen

Zu überprüfen, ob ein Programm mit zulässigen Eingaben funktioniert, ist nur die halbe Arbeit des Testens. In den folgenden Abschnitten besprechen wir einige wichtige Testsituationen, unter denen man jedes Programm testen sollte, bevor man ihm eine korrekte Funktionsweise bescheinigt.

Grenzfälle und Einschränkungen

Wenn Sie meinen, daß eine Routine mit einem bestimmten Bereich von Eingabewerten funktioniert, dann sollten Sie die Routine mit Daten testen, die in den Grenzbereichen der zulässigen Eingaben liegen. Wenn Sie zum Beispiel eine Routine haben, die eine Liste von 1 bis 20 Elementen darstellt, dann sollten Sie überprüfen, ob sie auch korrekt arbeitet, wenn genau ein Element in der Liste ist und exakt 20 Elemente in der Liste sind. Dabei kann sich dann einer der »eins-zuviel«- und »eins-zuwenig«-Fehler ergeben.

Falsche Eingabedaten

Sobald Sie der Ansicht sind, daß die Routine über den gesamten zulässigen Bereich von Eingabewerten funktioniert, sollten Sie feststellen, was die Routine macht, wenn unzulässige Daten eingegeben werden. Sie sollten überprüfen, ob die Routine die Eingabedaten als unzulässig zurückweist, auch wenn die Daten sehr nahe bei den zulässigen Daten liegen. Die Routine, die vorher die Werte von 1 bis 20 akzeptiert hat, sollte beispielsweise die Werte 0 und 21 zurückweisen.

Leere Eingabe

Das ist ein sowohl beim Testen als auch bei der Programmentwicklung häufig übersehener Bereich. Wenn Sie ein Programm schreiben, das eine vernünftige Voreinstellung aufweist, auch wenn Eingabedaten ausgelassen werden, erhöhen Sie die Benutzerfreundlichkeit Ihres Programms enorm.

Die Fehlersuche als Teil der Programmentwicklung

Wenn Sie mit dem Entwurf Ihres Programms anfangen, können Sie die Fehlersuche bereits planen. Einer der wesentlichen Punkte bei der Programmentwicklung betrifft den Grad der Datenüberprüfung in den einzelnen Teilen Ihres Programms.

Wenn Sie viele Überprüfungen vornehmen, erhalten Sie ein unverwüstliches Programm, das Sie über einen Fehler informiert, aber vernünftig weitermacht. Diese Art von Programmen kann sehr leicht zu debuggen sein, da Sie bereits von den Routinen selbst über unzulässige Daten informiert werden, bevor sich die Gefahr fortpflanzt.

Sie können auch ein Programm implementieren, das nur eine geringfügige oder gar keine Überprüfung der Ein-/Ausgabedaten vornimmt. Ihr Programm wird dann schneller und kompakter sein, aber mit falschen Eingabedaten oder einem kleinen Fehler kann man das Programm leicht aussteigen lassen. Diese Programme sind am schwierigsten zu debuggen, da sich ein kleines Problem während der Ausführung möglicherweise erst viel später auswirkt. Dadurch wird es schwierig, Ort und Stelle des ursprünglichen Fehlers zu finden.

Die meisten Programme enden als eine Mischform der beiden Methoden, wobei Eingabedaten von externen Quellen (wie zum Beispiel vom Anwender oder aus einer Datei) mit weitaus mehr Vorsicht behandelt werden als Daten, die intern von einer Routine an eine andere weitergegeben werden.

Beispiel für eine Debugger-Sitzung

In diesem Beispiel für die Fehlersuche mit einem Debugger setzen wir einige Methoden, die in den Kapiteln zuvor besprochen wurden, in die Praxis um. Das Programm TDDEMOB, das wir debuggen werden, ist eine bewußt fehlerhafte Version des Demonstrationsprogramms aus Kapitel 3 »Einführung in das Debuggen« (TDDEMO.C). Wie TDDEMO wurde auch TDDEMOB mit der *EasyWin* - Bibliothek von TurboC++ für Windows compiliert, damit die Ausgabe unter Windows angezeigt wird.

Die beiden Dateien für die Demonstration zur Fehlersuche sollten in Ihrem Arbeitsverzeichnis vorliegen. Wenn Sie C-Programmierer sind, benötigen Sie

TDDEMOB.C und TDDEMOB.EXE. (Das B in den Dateinamen steht für »buggy« [engl. fehlerhaft].)

Fehler suchen

Bevor wir mit dem Debuggen anfangen, wollen wir das fehlerhafte Programm einmal aufrufen und dann schauen, was daran falsch ist. Sie starten das Programm mit:

```
TDDEMOB
```

Sie werden nun nach einer Texteingabe gefragt. Geben Sie die beiden folgenden Zeilen ein:

```
eins zwei drei
vier fuenf sechs
```

Mit einer Leerzeile beenden Sie die Eingabe. TDDEMOB gibt anschließend eine Analyse Ihrer Eingaben aus:

```
Arguments
Enter a line (empty line to end): eins zwei drei
Enter a line (empty line to end): vier fuenf sechs
Enter a line (empty line to end):
Total number of letters = 8
Total number of lines = 6
Total word count = 2
Average number of words per line = 0.3333333
'E' occurs 2 times, 1 times at start of a word
'I' occurs 2 times, 0 times at start of a word
'N' occurs 1 times, 0 times at start of a word
'R' occurs 1 times, 0 times at start of a word
'S' occurs 1 times, 0 times at start of a word
'V' occurs 1 times, 1 times at start of a word
There are 2 words 4 characters long
```

Achten Sie darauf, daß falsche Zahlen für die Anzahl der Buchstaben, Zeilen und Wörter herauskommen. Die Häufigkeitstabellen für Buchstaben und Wörter scheinen ebenfalls auf einer falschen Buchstaben- und Wortanzahl zu beruhen. Dies ist eine überaus typische Situation: In dem Programm steckt mehr als nur ein Fehler. Das kommt im Frühstadium des Debuggens eines Programms sehr häufig vor.

Strategie festlegen

Ihre erste Aufgabe ist die Entscheidung, von welcher Seite Sie das Problem anpacken wollen. Eine gute Faustregel lautet, daß man das Problem als erstes

behandelt, das als erstes auftritt. In diesem Programm wird jede Eingabezeile in Worte aufgeteilt und anschließend analysiert. Nachdem schließlich alle Zeilen eingegeben wurden, werden die Tabellen ausgegeben. Da sowohl die Anzahl der Worte und Buchstaben als auch die Tabellen falsch sind, kann man Wetten darauf abschließen, daß bei der Aufteilung und beim Zählen der einzelnen Bestandteile etwas nicht in Ordnung ist.

Jetzt, nachdem Sie über das Problem einen Augenblick nachgedacht und einen groben Plan entworfen haben, ist der Zeitpunkt zum Debuggen gekommen. In diesem Fall ist die Strategie folgende: Überprüfen Sie die Routine *makeintowords* , ob die Zeile in mit NUL abgeschlossene Worte aufgeteilt wird, und untersuchen Sie, ob *analyzewords* die analysierten Bestandteile korrekt zusammenzählt.

Der Aufruf von TDW

Zum Starten der Debugger-Sitzung sollte das Editor-Fenster von Turbo C++ für Windows mit der Datei TDDEMOB aktiv sein. Ist dies der Fall, wird TDW nach Aufruf des Befehls *Run/Debugger* das fehlerhafte Demoprogramm laden und anschließend den Eingangsbildschirm darstellen. Wenn Sie die Beispielsitzung verlassen wollen, können Sie jederzeit Alt X eingeben, um zu Turbo C++ zurückzukehren. Wenn Sie sich hoffnungslos verstricken, können Sie das Demonstrationsprogramm in jeder Situation mit Strg F2 erneut laden und wieder von vorn anfangen. (Beachten Sie bitte, daß damit keine Breakpoints oder *Watches* -Variablen gelöscht werden.)

Da Sie als erstes die Routine *makeintowords* überprüfen wollten, müssen Sie das Programm bis zu dieser Routine laufen lassen und diese im Anschluß daran durchgehen. Sie können dabei von zwei Ansätzen ausgehen: entweder Schritt für Schritt mit *Step* durch die Routine *makeintowords* gehen und dabei sicherstellen, daß alles korrekt ist, oder Sie halten das Programm an, nachdem *makeintowords* alles erledigt hat und schauen dann nach, ob alles mit rechten Dingen zuging.

Da *makeintowords* eine ganz klar definierte Aufgabe hat und es nicht schwer zu ermitteln ist, ob die Routine funktioniert, wenn man sich den Ausgabepuffer anschaut, werden wir uns für letztere Methode entscheiden. Dazu positionieren wir den Cursor auf Zeile 42 und drücken F4, um das Programm bis zu dieser Zeile auszuführen. Es erscheint der Programmbildschirm und Sie sollten

```
eins zwei drei
```

eingeben und die Eingabe mit CR abschließen.

Untersuchungen

Sie haben das Programm nun in der Quellzeile hinter *makeintowords* angehalten. Sehen Sie sich den Inhalt von *buffer* einmal an, um festzustellen, ob alles

in Ordnung ist. Bewegen Sie den Cursor eine Zeile nach oben, stellen Sie ihn auf das Wort *buffer* und drücken Sie Alt F10 I (für Inspect), um das *Inspector*-Fenster zu öffnen und den Inhalt von *buffer* darzustellen. Verwenden Sie die Pfeiltasten, um durch die Elemente des Arrays zu wandern. Beachten Sie, daß *makeintowords* tatsächlich ein NUL-Byte hinter jedes Wort geschrieben hat. Das bedeutet, daß Sie das Programm etwas weiter laufen lassen und überprüfen müssen, ob *analyzewords* korrekt funktioniert. Zunächst müssen Sie das *Inspector*-Fenster mit Esc wieder schließen. Dann drücken Sie zweimal F7, um zum Anfang von *analyzewords* zu gelangen.

Breakpoints

Überprüfen Sie zunächst, ob *analyzewords* mit dem richtigen Pufferzeiger aufgerufen wird, indem Sie mit dem Cursor auf *bufp* setzen und Alt F10 I drücken. Sie sehen, daß *bufp* tatsächlich auf den mit NUL abgeschlossenen String *eins* zeigt. Drücken Sie Esc, um das *Inspector*-Fenster zu schließen. Da es anscheinend Probleme beim Zählen von Zeichen und Worten gibt, werden wir einen Breakpoint in den Programmstellen anbringen, in denen ein Zeichen und ein Wort gezählt werden:

1. Poitionieren Sie den Cursor auf Zeile 93 und drücken Sie F2, um einen Breakpoint zu setzen.

2. Setzen Sie in der Zeile 97 einen weiteren Breakpoint.

3. Schließlich setzen Sie noch einen Breakpoint in Zeile 99, um zu sehen, welchen Wert die Funktion als Anzahl der Zeichen zurückgibt.

Mehrere Breakpoints gleichzeitig zu setzen, ist eine gute Methode, um festzustellen, ob die Dinge in der richtigen Reihenfolge ablaufen. Damit kann man alle wichtigen Daten bei jedem Programmstop an einem Breakpoint untersuchen.

Das Watches-Fenster

Lassen Sie das Programm jetzt mit F9 laufen. Das Programm hält an, wenn es zum Breakpoint in Zeile 93 kommt. Nun wollen Sie sich den Wert von *charcount* anschauen. Da Sie diesen Wert jedesmal überprüfen wollen, wenn Sie zum Breakpoint kommen, ist das eine ideale Gelegenheit, um den *Watch* -Befehl zu verwenden und die Variable ins *Watches* -Fenster aufzunehmen. Bewegen Sie den Cursor zu *charcount* und drücken Sie Alt F10 W . Das *Watches* -Fenster am unteren Bildschirmrand stellt nun den aktuellen Wert 0 dar. Um sicherzustellen, daß die Zeichen korrekt gezählt werden, führen wir eine einzelne Zeile aus, indem wir F7 drücken. Das *Watches* -Fenster zeigt nun bei der Variablen *charcount* tatsächlich den Wert 1 an.

Das Dialogfenster Evaluate/Modify

Starten Sie das Programm wieder mit F9. Sie sind jetzt wieder zu Zeile 93 zurückgekommen, in der Sie das nächste Zeichen untersuchen können. Drücken Sie noch zweimal F9, um die letzten beiden Zeichen und das abschließende NUL-Zeichen des Worts zu lesen. *charcount* enthält nun den Wert 3, und das Array *wordcounts* sollte nun ein Wort weitergezählt werden. Bis jetzt ist alles klar. Drücken Sie F9, um mit der Verarbeitung des nächsten Worts im Puffer anzufangen. Aha! Hier ist etwas falsch.

Sie haben erwartet, daß das Programm in Zeile 93 anhält, um das nächste Wort zu verarbeiten, aber das war nicht der Fall. Das Programm wurde geradewegs bis zur Rücksprunganweisung ausgeführt. Die einzige Erklärung, wie das Programm zur Zeile 99 gelangen konnte, ist, daß die Bedingung der *while*-Schleife in Zeile 83 nicht mehr zutraf. Das bedeutet, daß **bufp != 0* falsch gewesen sein muß. Um das nachzuprüfen, markieren Sie in Zeile 83 den ganzen Ausdruck **bufp != 0*, indem Sie den Cursor auf das ***-Zeichen setzen, Einfg drücken und dann den Cursor zur *0* vor der schließenden Klammer ()) bewegen. Nun rufen Sie das Dialogfenster *Data/Evaluate Modify* auf und drücken CR, um die Auswahl mit dem *Eval*-Feld zu akzeptieren. Es ergibt sich tatsächlich der Wert 0. Drücken Sie zweimal Esc, um zum *Module*-Fenster zurückzukehren.

Es ist vollbracht!

Nun kommen wir zum Gedankensprung, mit dessen Hilfe Sie diesen Fehler beheben können. Der Grund, warum *bufp* auf eine 0 zeigt, liegt darin, daß *bufp* nach der *while*-Schleife in Zeile 86 auf das Ende des Worts zeigt. Um zum nächsten Wort zu kommen, muß *bufp* inkrementiert werden, damit er hinter das NUL-Byte zeigt, mit dem das vorige Wort abgeschlossen wurde. Dazu muß man vor der Zeile 97 die Anweisung *bufp++* einfügen. Sie könnten Ihr Programm mit dieser zusätzlichen Anweisung neu compilieren, aber mit TDW kann man zusätzliche Anweisungen in Ausdrücke »einflicken«, indem man eine besondere Art von Breakpoint verwendet.

Um das durchzuführen, müssen wir zunächst das Programm mit Strg F2 neu laden. Nun entfernen Sie alle Breakpoints, die Sie im letzten Durchlauf gesetzt haben, mit dem Befehl Alt B D. Gehen Sie zur Zeile 97 zurück, und setzen Sie den Breakpoint erneut mit der Taste F2. Setzen Sie diesen Breakpoint so, daß der Ausdruck *bufp++* jedesmal ausgeführt wird, wenn das Programm dort vorbeikommt:

1. Bei geöffnetem *Breakpoints*-Fenster und mit dem Cursor auf Zeile 97, drücken Sie Strg S, um das Dialogfenster *Breakpoint Options* zu öffnen.

2. Drücken Sie den Schalter *Change*, um das Dialogfenster *Conditions and Actions* zu öffnen.

3. Setzen Sie das Schaltfeld *Action* auf *Execute*.

4. Mit Tab gelangen Sie zu *Action expression*.

5. Geben Sie *bufp++* ein.

6. Drücken Sie zweimal CR, um zum *Breakpoints*-Fenster zurückzukehren. Im rechten Ausschnitt des Fensters wird nun die neue Aktion *Execute "bufp++"* angezeigt.

7. Drücken Sie Alt F3, um zum *Module*-Fenster zurückzukehren.

Starten Sie nun das Programm. Geben Sie wieder die beiden Zeilen ein:

```
eins zwei drei
vier fuenf sechs
```

Drücken Sie bei der dritten Eingabeanforderung CR und nach Abschluß des Programms Alt F5, um sich den Benutzerbildschirm anzuschauen.

Wie Sie sehen, hat sich die Angelegenheit merklich gebessert. Die Gesamtzahl der Worte und Zeilen scheint immer noch falsch zu sein, aber die Tabellen sind richtig. Drücken Sie eine Taste, um wieder zum *Module*-Fenster zurückzukehren.

Halten Sie das Programm in der Routine *printstatistics* an und kontrollieren Sie, ob die Routine die richtigen Werte für die Ausgabe erhält. Zunächst müssen Sie das Programm mit Strg F2 neu laden. Gehen Sie dann zur Zeile 104 und drücken Sie F4, um das Programm bis hierher auszuführen.

Geben Sie nun die beiden Eingabezeilen

```
eins zwei drei
vier fünf sechs
```

ein, und drücken Sie in der anschließenden Leerzeile CR.

Wieder zurück im *Module*-Fenster, bewegen Sie dann den Cursor auf das Argument *nlines* und drücken Sie Strg I, um diesen Wert unter die Lupe zu nehmen. Sie sehen den Wert 6, wo 2 stehen sollte.

Springen Sie nun zur Funktion *main* zurück und schauen Sie sich den Wert von *nlines* dort an. In *main* hat *nlines* den Wert 2, was durchaus richtig ist! Wenn Sie nun zur Zeile 45 gehen, werden Sie feststellen, daß *nwor*ds und *nlines* vertauscht wurden. Es gibt für den Compiler leider keine Möglichkeit, zu erkennen, daß Sie die umgekehrte Reihenfolge gemeint hatten.

Wenn Sie diese beiden Fehler ausbessern, läuft das Programm korrekt. Die Datei TDDEMO.EXE ist eine korrigierte Version, die Sie zum Vergleich heranziehen können, wenn Sie sich nicht sicher sind.

16. Ein ObjectWindows-Programm debuggen

Bei den Windows-Beispielprogrammen, die wir in diesem Kapitel verwenden, wurde zur Vereinfachung der Programmierung unter Windows die ObjectWindows-Programmbibliothek eingebunden.

Die Namen der Programme, die Sie hier verwenden, lauten TDODEMO und TDODEMOB (das *B* steht für engl.»*buggy*«, also fehlerhaft). TDODEMOB enthält mehrere Fehler, die wir nacheinander beim Durcharbeiten dieses Kapitels beheben werden.

Bevor Sie hier weitermachen, sollten Sie einmal TDODEMO von Windows aus starten und sich mit dessen Funktion und Aufbau vertraut machen. Zum Aufruf verwenden Sie entweder im Programm-Manager den Menübefehl *Datei/Ausf201hren*oder Sie fügen das Programm einer Programmgruppe hinzu und klicken dann auf das Symbol. (Wenn die Dateien TDODEMO und TDODEMOB nicht in Form von ausführbaren Dateien verfügbar sind, öffnen Sie die dazugehörigen Projektdateien und compilieren Sie beide Programme. Achten Sie darauf, daß die Debug-Informationen in die EXE-Dateien mit aufgenommen werden.)

Über das Programm

TDODEMO ist ein ObjectWindows-Programm, das es Ihnen erlaubt, mit Hilfe einer Maus in verschiedenen Farben auf dem Bildschirm zu zeichnen. Die linke Maustaste dient dabei zum Zeichnen der Linie, mit der rechten Maustaste können Sie den Bildschirm wieder löschen. Das Programm TDODEMO verfügt zudem über eine Menüleiste, mit der Sie eine von vier Farben zum Zeichnen auswählen können: Rot, Grün, Blau oder Schwarz.

Zum Zeichnen einer Linie halten Sie die linke Maustaste gedrückt, bewegen die Maus entsprechend und lassen die Maustaste wieder los. Das Programm löst diese Aufgabe ohne Probleme durch Verwendung der ObjectWindows-Bibliothek und der *dynamischen virtuellen Elementfunktionen* . Eine dynamische virtuelle Elementfunktion ist eine virtuelle Elementfunktion mit einer numerischen Kennung.

Da Turbo C++ für Windows den Windows-Botschaften Namen in Form von numerischen Konstanten gibt, können Sie den Namen einer Windows-Bot-

schaft als Index für eine dynamische virtuelle Elementfunktion verwenden. Damit kann ObjectWindows die Elementfunktion aufrufen, wenn das Fenster, für das diese Elementfunktion deklariert wurde, eine Botschaft erhält, die der numerischen Kennung der Funktion entspricht. Wenn es keine Elementfunktion mit passender numerischer Kennung gibt, ruft ObjectWindows die voreingestellte Windows-Funktion auf.

Um beispielsweise eine Funktion zu erzeugen, die auf eine WM_MOUSEMOVE-Botschaft antwortet, können Sie innerhalb eines Fensterobjekts eine Funktion folgender Art deklarieren:

```
virtual void WMMouseMove (RTMessage Msg) = [WM_FIRST+WM_MOUSEMOVE];
```

Wie Sie sehen, wird unmittelbar hinter der Funktionsdeklaration ein Gleichheitszeichen (=) verwendet, um der Funktion einen Ausdruck zuzuordnen, der in eckigen Klammern steht – in diesem Fall der Ausdruck WM_FIRST+WM_MOUSEMOVE. Sie addieren die ObjectWindows-Konstante WM_FIRST zur Konstante WM_MOUSEMOVE, um anzuzeigen, daß die Botschaftskonstante eine WM-typische Botschaft repräsentiert.

Der Typ *RTMessage* enthält die Windows-Prozedurparameter *wParam* und *lParam*. Diese Parameter enthalten oft zusätzliche Informationen über die Botschaft, etwa über die Position des Cursors.

In den nächsten Abschnitten wird erklärt, wie das Programm TDODEMOB funktioniert. Die darin enthaltenen Fehler werden dabei absichtlich übergangen, so daß Sie später die Möglichkeit haben, sie selbst zu entdecken. Es ist empfehlenswert, Turbo C++ für Windows zu starten und die Datei TDODEMPB.CPP zu öffnen, damit Sie die einzelnen Anweisungen parallel zur folgenden Beschreibung ansehen können.

Die Definition des Fenstertyps ScribbleWindow

Die Fensterklasse ScribbleWindow ist wie folgt definiert:

```
class ScribbleWindow : public TWindow
{
public:
  HDC HandleDC;       // Bildschirm-Kontext zum Zeichnen
  BOOL ButtonDown;    // Flag für »Linke Maustaste gedrückt«
  HPEN ThePen;        // Stift zum Zeichnen in Farbe
  ScribbleWindow(PTWindowsObject AParent, LPSTR ATitle);  ~ScribbleWindow();
  void GetWindowClass(WNDCLASS &AWndClass);
  virtual void WMLButtonDown(RTMessage Msg)=[WM_FIRST+WM_LBUTTONDOWN];
  virtual void WMLButtonUp(RTMessage Msg)=[WM_FIRST+WM_LBUTTONUP];
  virtual void WMMouseMove (RTMessage Msg)=[WM_FIRST+WM_MOUSEMOVE];
  virtual void WMRButtonDown(RTMessage Msg)=[WM_FIRST+WM_RBUTTONDOWN];
  virtual void SelectRedPen(RTMessage Msg)=[CM_FIRST+CM_RED];
  virtual void SelectGreenPen(RTMessage Msg)=[CM_FIRST+CM_GREEN];
```

```
    virtual void SelectBluePen(RTMessage Msg)=[CM_FIRST+CM_BLUE];
    virtual void SelectBlackPen(RTMessage Msg)=[CM_FIRST+CM_BLACK];
    virtual void SetupWindow();
};
```

Die Klasse *ScribbleWindow* definiert ein Fensterobjekt, das auf folgende Benutzereingaben reagiert:

- Bewegung der Maus;
- Linke Maustaste drücken und loslassen;
- Rechte Maustaste drücken;
- Farbauswahl und Position des Stifts.

Es gibt drei Datenelemente, *HandleDC*, *ButtonDown* und *ThePen*, in denen der Gerätekontext, der Status der Maustaste und der aktuelle Stift gespeichert sind.

ScribbleWindow
Der Konstruktor *ScribbleWindow* bindet das Menü an das Programm an und initialisiert das Datenelement *ButtonDown* mit FALSE sowie *ThePen* mit CM_BLACK.

GetWindowsClass
Die Elementfunktion *GetWindowClass* ruft die Standardfunktion *TWindow::GetWindowClass* auf, um das Fenster so einzurichten, daß es sich wie jedes andere *TWindow* verhält, und initialisiert anschließend das zum Programm gehörige Symbol.

WMLButtonDown
Drückt der Benutzer im *ScribbleWindow* -Fenster die linke Maustaste, um etwas zu zeichnen, erhält das Fenster eine WM_LBUTTONDWON-Botschaft. Dadurch wird ObjectWindows dazu veranlaßt, die Funktion *WMLButtonDown* aufzurufen (da dieser Funktion der Indexwert WM_FIRST+WM_LBUTTONDOWN zugeordnet wurde). *WMLButtonDown* bewegt den Stift an die aktuelle Position des Mauszeigers und setzt *ButtonDown*, um anzuzeigen, daß die Maustaste gedrückt ist. Anschließend wird *ThePen* als aktueller Gerätekontext ausgewählt. (Wir werden etwas später zeigen, daß diese Funktion dabei noch einige weitere Windows-Funktionen aufrufen sollte.)

WMLButtonUp
Wenn der Anwender mit dem Zeichnen fertig ist und die Maustaste losläßt, erhält das Fenster eine WM_LBUTTONUP-Botschaft, die ihrerseits wiederum ObjectWindows dazu veranlaßt, *WMLButtonUp* aufzurufen. Das Programm setzt die Programmvariable *ButtonDown* auf FALSE und gibt den mit diesem Fenster verbundenen Gerätekontext frei.

WMRButtonDown
Wenn der Anwender die rechte Maustaste drückt, um den Bildschirm zu

löschen, ruft ObjectWindows die Funktion *WMRButtonDown* auf, die die Windows-Funktion *UpdateWindow* aufruft. Durch den Aufruf dieser Funktion soll das Fenster gelöscht werden.

WMMouseMove
Wenn der Anwender einmal damit beginnt, die Maus über das Fenster zu bewegen, erhält das Fenster WM_MOUSEMOVE-Botschaften, wodurch ObjectWindows dazu veranlaßt wird, die Funktion *WMMouseMove* aufzurufen. Wenn der Anwender die linke Maustaste gedrückt hat, zieht das Programm bei jeder Bewegung der Maus eine Linie. Ist die Maustaste nicht gedrückt, passiert gar nichts.

Routinen für den Farbstift
Es gibt vier Funktionen, die durch Löschen des aktuellen Stiftes und Erzeugen eines neuen, andersfarbigen Stiftes eine neue Farbe zum Zeichnen bestimmen. Diese Funktionen unterscheiden sich nur durch die damit ausgewählte Farbe.

Das Programm erstellen

Um ein Anwendungsprogramm mit dem *ScribbleWindow* -Fenster zu erzeugen, muß eine Klasse abgeleitet werden, deren Basis die ObjectWindows-Klasse *TApplication* bildet. Zweck dieser Klasse namens *CScribbleApplication* ist es,

- die Funktion *InitMainWindow* so zu redeklarieren, daß die Anwendung ein Hauptfenster mit den Eigenschaften von *ScribbleWindow* erzeugen kann und

- einen Typ für das Objekt *MyApp* bereitzustellen, das für das Einrichten des Fensters und den Ablauf des Programms benötigt wird. (Das Fenster *MainWindow*, das weiter unten in diesem Kapitel vorkommt und dazu verwendet wird, Breakpoints für Fensterbotschaften zu setzen, ist ein Element von *MyApp*.)

Da Sie nun ungefähr wissen, wie das Programm funktionieren sollte, können Sie mit der Fehlersuche beginnen.

Das Programm debuggen

Falls Sie das nicht schon getan haben, starten Sie Turbo C++ für Windows und laden Sie die Projektdatei TDODEMOB.PRJ sowie die Programmdatei TDODEMOB.CPP. Starten Sie danach TDW mit *Start/Debugger* und drücken Sie dann F9, um das Demoprogramm in TDW auszuführen. Sie können den Mauszeiger bewegen und damit sogar Menübefehle auswählen; wenn Sie jedoch die linke Maustaste drücken, um mit dem Zeichnen zu beginnen, bricht das Programm ab und kehrt zu TDW zurück. Es erscheint die Fehlermeldung »Exception 13«.

(Diese Botschaft wird von TDW ausgegeben, wenn ein Programm einen UAE-Fehler, *Unrecoverable Application Error* , verursacht.)

Den ersten Fehler suchen

Wenn Sie Esc drücken, um das Fenster mit der Fehlermeldung zu schließen, wird das CPU-Fenster aktiviert. Dieses Fenster erscheint deshalb, weil Ihr Programm gerade Windows-Code ausführte, als der Fehler auftrat. Da das Programm nicht in das *Module*-Fenster zurückkehrte, haben Sie auch keinen Anhaltspunkt dafür, welcher Befehl Ihres Programms gerade ausgeführt wurde, als der Aufruf an Windows erfolgte, durch den der UAE-Fehler verursacht wurde.

Drücken Sie Alt F3 , um das CPU-Fenster zu schließen. (Sie werden hauptsächlich im *Module*-Fenster arbeiten).

Von welcher Funktion wurde Windows aufgerufen?

Da der unfreiwillige Ausstieg aus dem Programm beim Drücken der linken Maustaste erfolgte, liegt das Problem vermutlich in der Funktion *WMLButtonDown*. Es gibt allerdings auch noch eine andere Methode, derer Sie sich bedienen können, um die Stelle herauszufinden, an der Ihr Programm abgebrochen wurde – die Untersuchung des Stack.

Für diese Untersuchung aktivieren Sie das Stack-Fenster (*View/Stack*), folgen der Liste mit den hexadezimalen Angaben nach unten, bis Sie zu einer Zeile gelangen, die auf eine Routine Ihres Programms hinweist (der Name der Routine wird in ASCII-Zeichen ausgegeben). Das ist die Zeile, von der aus der Windows-Kernel aufgerufen wurde.

Dem Stack-Fenster können Sie entnehmen, daß der Fehler tatsächlich in der Routine *WMLButtonDown* zu suchen ist. Um im *Module*-Fenster an diese Routine heranzukommen, klicken Sie zuerst in dieses Fenster. Drücken Sie anschließend Strg S , geben Sie `WMLButtonDown` ein und drücken Sie CR, um diese Routine zu finden. Wenn die Meldung `Search expression not found` erscheint, positionieren Sie den Cursor auf den Anfang der Datei und drücken die Tastenkombination Strg N , um die Suche zu wiederholen (in TDW kann eine Suche nur von der Cursor-Position bis zum Dateiende durchgeführt werden). Unter Umständen müssen Sie mehrmals Strg N drücken, bis Sie die eigentliche Funktion gefunden haben.

Debuggen von WMLButtonDown

WMLButtonDown übernimmt als Parameter eine Variable vom Typ *RTMessage* und entnimmt dieser Botschaft die Position des Mauszeigers. *WMLButtonDown* ruft dann die Windows-Funktionen *MoveTo* und *SelectObject* auf, um den Zeichenstift im Fenster zu positionieren und ihn als aktuelles Symbol zum Zeichnen auszuwählen.

Da Sie den Namen dieser Routine im Stack-Fenster gesehen haben, muß einer dieser beiden Windows-Aufrufe fehlerhaft sein. Um herauszufinden, welcher das ist, lassen Sie das Programm bis zum Anfang dieser Funktion ablaufen und

arbeiten Sie die Funktion dann im Einzelschrittmodus ab. Auf diese Weise sehen Sie, welcher Aufruf den UAE-Fehler verursacht.

Wenn der Cursor in der ersten Zeile von *WMLButtonDown* steht, laden Sie durch Betätigen von Strg F2 von neuem das Programm TDODEMOB; anschließend drücken Sie F4, um das Programm bis zu diesem Punkt hin auszuführen. Wenn das *ScribbleWindow* -Fenster erscheint, drücken Sie die linke Maustaste, damit die Programmausführung abgebrochen wird und damit die Steuerung an TDW zurückkehrt. (Möglicherweise müssen Sie eine Taste drücken, damit Windows die Mausbotschaften freigibt.) Diesmal erscheint keine UAE-Fehlermeldung (besser gesagt: noch keine), denn bisher wurde nur der Aufruf an die Funktion *WMLButtonDown* aus TDODEMOB abgearbeitet.

Führen Sie mit Hilfe der Taste F7 das Programm im Einzelschrittmodus aus. Wenn Sie durch Betätigen von F7 *MoveTo* auslösen, wird die Fehlermeldung `Exception 13` ausgegeben. Das Problem muß also im Aufruf von *MoveTo* liegen.

Debuggen von MoveTo

Der Funktion *MoveTo* werden drei Parameter übergeben: Das Handle des Stiftes sowie die X- und die Y-Koordinate der aktuellen Position des Mauszeigers. Diese Koordinaten sind dem Parameter *Msg* entnommen, dem sie seinerseits von Windows übermittelt wurden. Wenn das Programm nicht den falschen Teil dieser Botschaft erwischt (das ist nicht der Fall), sollten diese beiden Parameter in Ordnung sein.

Dann ist der Fehler wohl eher bei *HandleDC* zu suchen, dem Handle des Gerätekontexts.

Da hier zwei UEAs aufgetreten sind, ist es am sichersten, TDW zu verlassen und Windows zu beenden, bevor Sie weitermachen.

Den Fehler lokalisieren

Wenn *HandleDC* die Ursache des UAE-Fehlers war, wurde das Handle entweder falsch gesetzt oder nie initialisiert. Tatsächlich – das Handle wurde nicht initialisiert. Das Programm hätte das Handle durch Initialisieren eines Bildschirmkontexts mit folgendem Windows-Funktionsaufruf setzten sollen:

```
HandleDC = GetDC(HWindos);
```

Der folgende Code zeigt *WMLButtonDown* mit der zusätzlichen Initialisierungsanweisung für den Kontext:

```
void ScribbleWindows::WMLButtonDown(RTMessage Msg)
{
  if ( !ButtonDown )
  {
    ButtonDown = TRUE;      // Setzt den Zustand der Maustaste
                            // auf »gedrückt«; wenn die Maus bewegt
                            // wegt wird, wird eine Linie gezeichnet.
    HandleDC = GetDC(HWindow);

                            // Bildschirmkontext für
                            // Zeichnen erzeugen.
    MoveTo(HandleDC, Msg.LP.LO, Msg.LP.HI);
```

```
                                // Zeichenpunkt an die Stelle
                                // verschieben, an der die
                                // Maustaste gedrückt wurde.
      SelectObject(HandleDC, ThePen);
                                // Stift als Bildschirmkontext.
    }
  }
```

Überprüfen der Korrektur

Fügen Sie in Turbo C++ für Windows die Anweisung zum Initialisieren des Kontexts in *ScribbleWindow::WMLButtonDown* ein. Compilieren Sie dann das Projekt zusammen mit den Debug-Informationen (verwenden Sie dazu *Compile/Build all*).

Für den Fall, daß das Programm noch mehr Fehler enthält, laden Sie es wieder in TDW und drücken anschließend bei Erscheinen des *Module*-Fensters die Taste F9.

Wenn Sie nun mit dem Stift zeichnen, sehen Sie, daß Ihre Zeichnung in schwarzer Farbe erscheint. Versuchen Sie auch andere Farben über das Menü zu wählen. Rot, Grün und Blau funktionieren problemlos. Wenn Sie jedoch versuchen, die Farbe wieder auf Schwarz zurückzustellen, wird sich die Farbe des Stiftes nicht ändern. Ein zweiter Fehler!

Den Farbzuordnungsfehler lokalisieren

Sehr wahrscheinlich liegt dieser weitere Fehler in der *ScribbleWindow* -Funktion, die den schwarzen Stift erzeugen soll, also in *SelectBlackPen*. Verlassen Sie TDODEMOB und drücken Sie anschließend Strg F2, um das Programm in TDW erneut zu laden. Setzen Sie bei der öffnenden geschweiften Klammer von *ScribbleWindow::SelectBlackPen* einen Breakpoint, starten Sie dann das Programm erneut und wählen Sie *Pen/Black*. (Es kann sein, daß Sie eine Taste drücken müssen, bevor Windows eine Mausbotschaft ausgibt.) TDW hätte die Ausführung des Programms bei dem Breakpoint anhalten müssen. Da dies nicht geschehen ist, muß der Fehler anderswo liegen.

Es sieht so aus, als würde *SelectBlackPen* überhaupt nicht aufgerufen. Da diese Routine über eine dynamische Tabelle virtueller Elementfunktionen (engl. *Dynamic Dispatched Virtual Table*, DDVT) aufgerufen wird, ist dieser Funktion in der DDVT möglicherweise ein falscher Indexwert zugeordnet.

Breakpoints für Fensterbotschaften setzen

Wenn ein Anwender einen Menübefehl auswählt, sendet Windows eine WM_COMMAND-Botschaft an das dazugehörige Fenster. Der Parameter *wParam* dieser Botschaft enthält den Indexwert des ausgewählten Menübefehls. Wenn ein ObjectWindows-Fenster eine WM_COMMAND-Botschaft erhält, durchsucht es die Indexwerte der dynamischen Tabelle virtueller Elementfunktionen des Fensterobjekts nach dem Wert von CM-FIRST+*wParam*. *SelectBlack-*

Pen hat den Indexwert CM_FIRST+CM_BLACK, wobei CM_BLACK den Wert 104 repräsentiert.

Um herauszufinden, welchen Wert der Parameter *wParam* besitzt, wenn der Anwender den Menübefehl *Pen/Black* wählt, müssen Sie TDW anweisen, die Ausführung zu unterbrechen, wenn eine WM_COMMAND-Botschaft eintrifft. Sie können dazu das Programm ablaufen lassen, die Menüauswahl treffen und dann *wParam* überprüfen, um zu sehen, ob ihr Wert dem Wert der Konstanten CM_BLACK entspricht.

Bevor Sie den Breakpoint setzen können, müssen Sie die Steuerung wieder zu TDW zurückgeben. Beenden Sie TDODEMOB und laden Sie es anschließend, wenn das *Module*-Fenster wieder aktiv ist, mit Strg F2 erneut. Der nächste Schritt besteht nun darin, einen Fensterbotschafts-Breakpoint zu setzten. Dazu gibt es zwei Möglichkeiten, je nachdem, ob Sie mit TDWINST die Unterstützung von ObjectWindows-Botschaften aktiviert haben oder nicht.

Breakpoints für Fensterbotschaften unter ObjectWindows werden nicht per Voreinstellung unterstützt. Sie können somit ein Fensterobjekt, das Sie in Ihrem Programm deklariert haben, nicht dazu verwenden, um einen Breakpoint für eine Fensterbotschaft zu setzen; stattdessen müssen Sie das Fenster-Handle verwenden. Wenn Sie ein Fensterobjekt verwenden möchten (was einfacher ist, das Debuggen jedoch verlangsamen könnte, falls Sie einen Breakpoint für eine Fensterbotschaft gesetzt haben), müssen Sie TDWINST starten, *Options/Source Debugging* auswählen und anschließend *OWL Windows Message* markieren. (Nähere Informationen über TDWINST finden Sie in Anhang C, »Die Installation von TDW«.)

Fensterbotschaften-Breakpoints über das Fenster-Handle setzen

Haben Sie bei der Installation von TDW die Unterstützung von ObjektWindows nicht aktiviert, dann müssen Sie, wenn Sie einen Breakpoint für eine Fensterbotschaft setzen möchten, das zugehörige Fenster-Handle verwenden. Da beim Einrichten eines Fensters der Großteil der Arbeit von ObjektWindows erledigt wird, kann es einige Zeit dauern, bis Sie auf das Handle zugreifen können.

Das Einrichten des Fensters wird von der Funktion *InitMainWindow* erledigt, das Fenster-Handle wird jedoch erst später gesetzt. Damit TDW das Programm TDODEMOB unterbrechen und die Steuerung wieder wieder übernehmen kann, könnten Sie in einer der Funktionen zur Bearbeitung von Mausklicks, einen Breakpoint setzen (etwa in *WMLButtonDown*), TDODEMOB starten und den Breakpoint anschließend mit einem Mausklick auslösen. (Wenn Ihr Breakpoint z.B. in *WMLButtonDown* gesetzt ist, könnten Sie etwa die linke Maustaste betätigen.)

Eine andere Vorgehensweise besteht darin, die ObjectWindows-Funktion, die das Handle initialisiert, also *SetupWindows* , zu überschreiben; diese Funktion wird in TDODEMOB als virtuelle Funktion redeklariert und folgendermaßen definiert:

```
void ScribbleWindow::SetupWindow()
{
  TWindow::SetupWindow();
}
```

Für diese Vorgehensweise positionieren Sie den Cursor auf die abschließende geschweifte Klammer der Funktion *SetupWindow* und drücken dann F4, um das Programm bis zu dieser Stelle auszuführen.

Gleichgültig, welche der beiden Methoden Sie verwenden: Wenn TDW die Steuerung zurückerlangt, müssen Sie folgende Schritte durchführen, um einen Fensterbotschafts-Breakpoint auf die WM_COMMAND-Botschaft zu setzen:

1. Wählen Sie *Data/Inspect* und untersuchen Sie das Fensterobjekt *MainWindow*. Da es außerhalb des aktuellen Gültigkeitsbereichs liegt, müssen Sie den Gültigkeitsbereich auf folgende Weise vorgeben:

```
WinMain#MyApp.MainWindow
```

MainWindow ist ein Element von *MyApp*, da *MyApp* vom Typ *CScribbleApplication* ist. *CScribbleApplication* wiederum ist abgeleitet von der Klasse *TApplication*, die *MainWindow* als Datenelement enthält.

2. Vergrößern Sie das *Inspector*-Fenster, so daß Sie die Datenelemente im oberen Ausschnitt sehen. *HWindow* ist das Datenelement, welches das zu diesem Fenster gehörende Handle enthält.

3. Setzen Sie den Cursor auf *HWindow*; drücken Sie dann Umschalt F3, um das Datenelement in die Zwischenablage zu kopieren.

4. Wählen Sie *View/Windows Messages*, um das Dialogfenster *Windows Messages* zu öffnen.

5. Drücken Sie Strg A im linken oberen Ausschnitt, um das Dialogfenster *Add* anzuzeigen. Wählen Sie das Feld *Handle*, und aktivieren Sie dann das Eingabefeld.

6. Drücken Sie Umschalt F4, um die Zwischenablage anzuzeigen. Setzen Sie den Cursor auf *HWindow*, aktivieren Sie das Feld *Contents* (um den Inhalt von *HWindows*, also den Wert des Handle zu kopieren), und wählen Sie dann *OK*, um den Wert des Handle in das Eingabefeld zu übertragen.

7. Fügen Sie nun im Eingabefeld am Anfang des Handle-Wertes `0x` ein, um anzuzeigen, daß es sich um einen hexadezimalen Wert handelt, und drücken Sie anschließend CR.

8. Wechseln Sie in den rechten oberen Ausschnitt und geben Sie `WM_COMMAND` ein. Während der Eingabe erscheint das Dialogfenster *Set Message Filter*.

9. Stellen Sie *Break* als Aktion für diese Botschaft ein und drücken Sie dann CR, um für diese Botschaft einen Breakpoint zu setzen.

Jedesmal, wenn Sie von jetzt an eine Menüauswahl treffen, übergibt das Programm die Steuerung wieder an TDW, denn beim Auswählen eines Menübefehls wird eine WM_COMMAND-Botschaft erzeugt.

Fensterbotschafts-Breakpoints über ein Fensterobjekt setzen

Wenn Sie mit TDWINST die Unterstützung von Fensterbotschafts-Breakpoints von ObjectWindows aktiviert haben, können Sie das Fensterobjekt *MainWindow* dazu verwenden, einen Breakpoint für Fensterbotschaften zu setzen.

1. Positionieren Sie den Cursor auf die abschließende geschweifte Klammer der Funktion *InitMainWindow* und drücken Sie dann F4, um das Programm bis zu dieser Stelle auszuführen.

2. Wenn anschließend wieder das *Module*-Fenster erscheint, öffnen Sie das Dialogfenster *OWL Windows Messages* mit *View/Windows Masseges*.

3. Geben Sie im oberen linken Ausschnitt *MainWindows* ein und drücken Sie CR

4. Geben Sie nun im rechten oberen Ausschnitt `WM_COMMAND` ein und wählen Sie dann den Schalter *Break*. Mit CR bestätigen Sie anschließend das Setzen des Breakpoints.

Von nun an gibt das Programm die Steuerung jedesmal an TDW zurück, wenn Sie ein Menü auswählen, denn dadurch wird eine WM_COMMAND-Botschaft erzeugt.

wParam untersuchen

Sie können die Ausführung des Programms durch Drücken der Taste F9 wieder aufnehmen.

Wählen Sie aus dem Menü nun *Pen/Black Pen*. Sobald Sie den schwarzen Stift gewählt haben, unterbricht TDW die Ausführung des Programms und öffnet das CPU-Fenster, wodurch angezeigt wird, daß das Programm zu dem Zeitpunkt, zu dem der Breakpoint ausgelöst wurde, Code des Windows-Kernel ausgeführt hat. Schließen Sie das CPU-Fenster mit Alt F3.

Sofern erforderlich, aktivieren Sie nun erneut das Fenster *Windows Messages*. Bringen Sie das Fenster auf volle Größe, damit Sie im unteren Ausschnitt die ganze Botschaft lesen können. Sie erkennen, daß das Fenster eine WM_COMMAND-Botschaft mit dem Wert 204 (00CCh) in *wParam* erhielt. Der Grund dafür, daß die virtuelle Funktion nicht aufgerufen wurde, war also, daß die Anwendung nach einem Indexwert von CM_FIRST+204 gesucht hat, der aktuelle Wert jedoch CM_FIRST+104 beträgt.

Der Wert 204 stammt aus der Menüdefinition in der Datei TDODEMO.RC. Dieser Fehler hätte vermieden werden können, wenn für das Menü die gleichen Indexwerte wie in der Header-Datei verwendet und die Headerdatei per #INCLUDE-Anweisung am Anfang der .RC-Datei aufgenommen worden wäre. Stattdessen wurden in dieser Menüdefinition Indexwerte in Form von Integer-Zahlen verwendet; in diesem Fall liegt die Verantwortung für die korrekten Werte beim Programmierer.

Wenn Sie die Datei TDODEMOB.H editieren und CM_BLACK mit dem Wert 204 definieren, müßte die Auswahl des schwarzen Stifts korrekt funktionieren. Nach dieser Änderung lauten die Konstantendeklarationen in TDODEMOB.H folgendermaßen:

```
#define PenWidth    1
#define MenuID      100
#define IconID      100
#define CM_RED      101
#define CM_GREEN    102
#define CM_BLUE     103
#define CM_BLACK    204
```

Überprüfen der Korrektur

Lassen Sie TDODEMOB bis zum Programmende ablaufen und verlassen Sie dann das Programm. Verlassen Sie anschließend auch TDW. Laden Sie in Turbo C++ für Windows die Header-Datei TDODEMOB.H und ändern Sie die Definition der Konstanten CM_BLACK. Compilieren Sie das Projekt anschließend neu und starten Sie das Programm unter TDW.

Wenn Sie nun im *ScribbleWindow* -Fenster zeichnen, tritt möglicherweise ein neues Problem auf. Bewegen Sie während des Zeichnens den Mauszeiger aus dem Fenster heraus und anschließend an einer anderen Stelle wieder in das Fenster hinein. Sie werden feststellen, daß das Programm eine gerade Linie gezeichnet hat, die den Punkt, an dem der Zeichenstift das Fenster verlassen hat, und den Punkt, an der er wieder zurückkehrt, miteinander verbindet. Das Programm sollte jedoch mit dem Zeichnen aufhören, wenn der Zeichenstift außerhalb des Fensterbereiches ist und wieder neu anfangen, wenn der Zeichenstift in das Fenster eintritt. Sie haben also einen weiteren Fehler gefunden.

Lokalisieren des Fensterbereichsfehlers

Dieser Fehler ist bei den Fensterbotschaften zu suchen, die an dieses Fenster übermittelt werden. Verlassen Sie das Programm und laden Sie TDODEMOB.CPP in das *Module*-Fenster von TDW.

Protokollieren der Fensterbotschaften

Je nachdem, ob Sie Fensterbotschaften unter ObjectWindows aktiviert haben, verwenden Sie eine der beiden möglichen Methoden zum Initialisieren von Fenstern, die oben beschrieben wurden. Geben Sie dann im oberen linken Abschnitt des Fensters *Windows Messages* (entweder über das Fensterobjekt oder über das Handle des Fensters) an, für welches Fenster die Botschaften aufgezeichnet werden sollen.

Als nächstes setzen Sie den Cursor in den oberen rechten Ausschnitt. Dann setzen Sie bei WM_LBUTTONUP einen Breakpoint für eine Botschaft, damit die Steuerung nach Beenden des Zeichnens wieder an TDW übergeht.

Vermutlich wollen Sie alle Botschaften mitverfolgen, die Sie erhalten können; das Setzen eines Breakpoint bei WM_LBUTTONUP hat jedoch die Einstellung *Log All Messages* gelöscht. Um diese Einstellung wiederherzustellen, drükken Sie Strg A um das Dialogfenster *Set Message Filter* zu öffnen und wählen anschließend darin die Klasse *Log All Messages* .

Ermitteln des Fehlers

Nehmen Sie die Ausführung von TDODEMOB mit F9 wieder auf. Beginnen Sie mit dem Zeichnen, bewegen Sie den Mauszeiger aus der für das Zeichnen vorgesehenen Fläche wieder heraus, bewegen Sie ihn etwas hin und her und bewegen Sie ihn dann wieder (an einer anderen Stelle) in die Zeichenfläche hinein. Lassen Sie dann die linke Maustaste los, damit die Steuerung wieder an TDW zurückgeht.

Bevor Sie sich nun wieder dem Fenster *Windows Messages* zuwenden, vergrößern Sie es auf die volle Größe (drücken Sie dazu die Taste F5), so daß Sie mehr Botschaften sehen können. Wenn Sie nun den unteren Ausschnitt des Fensters *Windows Messages* betrachten, sehen Sie mehrere WM_NCHITTEST- und WM_SETCURSOR-Botschaften.

Suchen Sie die Botschaft WM_LBUTTONDOWN, denn hier haben Sie mit dem Zeichnen begonnen. Nach dieser Botschaft finden Sie einige Serien von WM_NCHITTEST-, WM_SETCURSOR- und WM_MOUSEMOVE-Botschaften; daran anschließend sehen Sie mehrere Serien von WM_NCHITTEST-, WM_SETCURSOR- und WM_NCMOUSEMOVE-Botschaften, schließlich noch eine WM_LBUTTONUP-Botschaft.

Eine WM_MOUSEMOVE-Botschaft kann nur im Client-Bereich des Programms auftreten. Eine solche Botschaft wird ausgegeben, wenn der Mauszeiger aus dem Client-Bereich des Programms hinausbewegt wird.

Jetzt ist klar, worin der Fehler besteht. Das Programm zeichnet eine Linie von der Stelle der letzten WM_MOUSEMOVE-Botschaft bis zu der Stelle der aktuellen WM_MOUSEMOVE-Botschaft. Wenn der Mauszeiger den Client-Bereich verläßt, erhält das Programm keine WM_MOUSEMOVE-Botschaften. Kehrt also der Mauszeiger in den Client-Bereich zurück, bildet die Stelle, an der der Mauszeiger den Client-Bereich verlassen hat, den Anfangspunkt einer Linie, und das Programm zeichnet stur eine Linie von diesem Punkt zum nächsten.

Den Fehler beheben

Eine mögliche Lösung zur Behebung dieses Fehlers besteht darin, genau zu bestimmen, wann der Mauszeiger den Client-Bereich verläßt, so daß das Programm die letzte Position des Mauszeigers ignorieren kann und mit dem Zeichnen erst wieder fortfährt, wenn sich der Mauszeiger wieder im Client-Bereich befindet. Dazu wären einige logische Arbeitsschritte notwendig, um festzustellen, wann der Mauszeiger den Client-Bereich verläßt und wann er wieder in ihn eindringt. Glücklicherweise gibt es eine einfachere Lösung.

Die Windows-Funktion *SetCapture* erfüllt die notwendigen Anforderungen. Diese Funktion veranlaßt Windows dazu, alle mausbezogenen Botschaften an das dafür vorgesehene Fenster zu senden bis das Programm die Routine *ReleaseCapture* aufruft und damit bewirkt, daß das Fenster (anstelle der WM_NCMOUSEMOVE-Botschaften) die Botschaft WM_MOUSEMOVE erhält, wenn der Mauszeiger aus dem Client-Bereich herausbewegt wurde.

Wenn Sie *SetCapture* in *ScribbleWindow::WMLButtonDown* aufnehmen und *ReleaseCapture* in *WMLButtonUp*, zeichnet *WMMouseMove* tatsächlich auch außerhalb des Fensters, wenn der Mauszeiger in diesen Bereich bewegt wird. Allerdings schneidet Windows alle Linien, die außerhalb des Client-Bereichs gezeichnet werden, ab und erzeugt auf diese Weise den gewünschten Effekt.

Diese Änderungen werden im folgenden Listing zusammengefaßt:

```
void ScribbleWindow::WMLButtonDown(RTMessage Msg)
{
  if ( !ButtonDown )
  {
    ButtonDown = TRUE;   // Definiert die Maustaste als »gedrückt«;
                         // wenn also eine Mausbewegung stattfindet,
                         // wird eine Linie gezeichnet.
    SetCapture(HWindow); // Windows wird angewiesen, alle Maus-
                         // Botschaften an das Fenster zu senden.
                         // Die Funktion WMLButtonUp hebt diesen
                         // Zustand wieder auf.
    HandleDC = GetDC(HWindow);
                         // Erzeugt Bildschirmkontext zum Zeichnen
    MoveTo(HandleDC, Msg.LP.Lo, Msg.LP.Hi);
                         // Setzt zu zeichnenden Punkt an die
                         // Stelle, an der die Maustaste gedrückt
                         // wurde.
    SelectObject(HandleDC, ThePen);
                         // Bestimmt den Stift als
                         // Bildschirmkontext.
  }
}

void ScribbleWindow::WMLButtonUp(RTMessage)
{
  if ( ButtonDown )
  {
    ReleaseCapture();
    ReleaseDC(HWindow, HandleDC);
    ButtonDown = FALSE;
  }
}
```

Überprüfung der Korrektur

Verlassen Sie jetzt TDODEMOB und anschließend TDW. Ändern Sie in Turbo C++ für Windows die beiden Routinen, compilieren anschließend das Programm und starten Sie es. Wenn Sie nun wieder im dafür vorgesehenen Fenster zeichnen, funktioniert alles wie vorgesehen. Versuchen Sie allerdings, den Bildschirm durch Betätigen der rechten Maustaste zu löschen, geschieht nichts. Sie haben einen weiteren Fehler gefunden.

Den Fehler beim Bildschirmlöschen lokalisieren

Für die rechte Maustaste ist die Funktion *WMRButtonDown* zuständig; der Fehler hat also vermutlich etwas mit dieser Funktion zu tun. Entweder wird *WMRButtonDown* nicht aufgerufen oder die Funktion ist fehlerhaft.

Verlassen Sie TDODEMOB und laden Sie anschließend TDODEMOB in TDW. Um das Programm bis zur Funktion *WMRButtonDown*, der wahrscheinlichen Fehlerquelle, auszuführen, drücken Sie Alt F9 und geben dann `ScribbleWindow::WMRButtonDown` ein. Zeichnen Sie nun etwas in das Fenster und drücken Sie danach die rechte Maustaste. TDW hält das Programm am Anfang der Funktion *WMRButtonDown* an; dadurch wissen Sie, daß diese Funktion aufgerufen wurde.

Führen Sie die Funktion *WMRButtonDown* mit F7 nun schrittweise bis zum Aufruf von *UpdateWindow* aus. (Betätigen Sie jetzt noch nicht die Taste F7!) Der einzige Parameter dieser Funktion ist *HWindow*. Sie können davon ausgehen, das *HWindow* korrekt gesetzt wurde, da auch andere Funktionen damit ordnungsgemäß arbeiten. Da Sie hier offensichtlich keinen Fehler finden, sollten Sie prüfen, ob die Botschaft WM_PAINT (sie müßte durch den Aufruf von *UpdateWindow* an das Fenster gesendet werden) tatsächlich dort eintrifft.

Mittlerweile wissen Sie wahrscheinlich schon sehr genau, wie ein Botschafts-Breakpoint auf auf WM_PAINT gesetzt wird. Falls Sie dennoch Fragen haben, arbeiten Sie die entsprechenden Erläuterungen weiter oben noch einmal durch. Zusätzlich zum Setzen eines Botschafts-Breakpoint auf WM_PAINT können Sie auch ein Protokoll der WM_PAINT-Botschaften anlegen. (Geben Sie dazu im Dialogfenster *Set Message Filter* ein weiteres mal WM_PAINT ein, aktvieren Sie jetzt jedoch das *Log*-Feld.)

Nachdem der Botschafts-Breakpoint gesetzt ist, drücken Sie F7, um den Aufruf von *UpdateWindows* auszuführen. Die Tatsache, daß die Programmausführung nicht unterbrochen wird, läßt nur eine Schlußfolgerung zu: WM_PAINT wird nie an das Fenster gesendet.

Sie können sich davon überzeugen, daß keine WM_PAINT-Botschaften an das Fenster gesendet werden, indem Sie die rechte Maustaste drücken, um bei *WMRButtonDown* die Steuerung an TDW zu übergeben. Überprüfen Sie dann den unteren Ausschnitt des Dialogfensters *View/Windows Messages*. Es gibt hier keine WM_PAINT-Botschaften. Aus irgendwelchen Gründen arbeitet der Aufruf von *UpdateWindow* nicht korrekt.

Die Analyse der Fehlerquelle

Um diesen Fehler zu beheben ist es notwendig, sich etwas näher mit der Art und Weise zu befassen, wie Windows die Funktion *UpdateWindow* handhabt. Wenn ein Programm diese Funktion aufruft, überprüft Windows zunächst, ob ein Teil des Fensters ungültig ist und neu aufgebaut werden muß. Ist dies der Fall, schickt Windows eine WM_PAINT-Botschaft an das Fenster. Ist dies nicht der Fall, besteht kein Grund, Systemressourcen mit unnötigen Botschaften zu belasten. Wie aber findet Windows heraus, daß das Fenster neu aufgebaut werden muß?

Ein Programm informiert Windows darüber, daß zumindest ein Teil eines Fensters ungültig ist, indem es entweder *InvalidRect* oder *InvalidateRgn* aufruft.

Diese beiden Funktionen bestimmen den erneuerungsbedürftigen Bereich des Fensters und bewirken, daß Windows zum Erneuern des Fensters eine WM_PAINT-Botschaft an das Fenster sendet.

Eigentlich könnten Sie den Aufruf von *UpdateWindow* durch *InvalideRect* ersetzen. Windows weist jedoch der WM_PAINT-Botschaft, die es als Reaktion auf eine dieser beiden Funktionsaufrufe absendet, eine geringe Priorität zu. Wenn Sie also sicherstellen möchten, daß das Fenster unverzüglich erneuert werden soll, müssen Sie den Aufruf *UpdateWindow* beibehalten.

Den Fehler beheben

Unser Problem kann gelöst werden, indem ein Aufruf an *InvalidateRect* in *WMRButtonDown* hinzugefügt wird. *InvalidateRect* benötigt drei Parameter: ein Fenster-Handle, einem Zeiger auf einen Rechteckbereich, der erneuert werden soll, und einem Booleschen Parameter, der festlegt, ob das Rechteck gelöscht werden soll. Für den Zeiger auf das Rechteck können Sie NULL übergeben. Damit weisen Sie Windows an, daß das ganze Fenster als Rechteckbereich verwendet werden soll. Das folgende Listing zeigt *WMRButtonDown* mit dem neuen Funktionsaufruf:

```
void ScribbleWindow::WMRButtonDown(RTMessage)
{
  InvalidateRect(HWindow, NULL, TRUE);
  UpdateWindow(HWindow);
}
```

Die Korrektur überprüfen

Beenden Sie TDODEMOB und verlassen Sie anschließend auch TDW. Führen Sie in Turbo C++ für Windows die Änderungen für *WMRButtonDown* durch, compilieren dann das Programm und starten Sie es. Wenn Sie nun die rechte Maustaste drücken, nachdem Sie etwas gezeichnet haben, wird der Fensterinhalt tatsächlich gelöscht.

Damit haben wir endlich alle Fehler gefunden, lokalisiert und behoben. Das Programm arbeitet nun einwandfrei.

A. Kommandozeilen-optionen

Beim Aufruf von TDW mit Hilfe des Befehls *Datei/Ausführen* im Programm-Manager können Sie zusätzliche Optionen in der Kommandozeile angeben. Die allgemeine Syntax des Aufrufs lautet:

```
tdw [ Optionen ] [ Programmname [ Programmargumente ] ]
```

Optionale Eingaben stehen in eckigen Klammern. Ein Minuszeichen hinter einer Option schaltet die entsprechende Option ab.

Tabelle A.1 Die Kommandozeilenoptionen von TDW

Option	Beschreibung
-c*Dateiname*	Konfigurationsdatei.
-do	Debugger verwendet den zweiten Bildschirm.
-ds	Bildschirmdarstellung von Programm und Debugger wird in den Programmspeicher ausgelagert).
-h, -?	Hilfsinformationen für alle Kommandozeilenoptionen ausgeben.
-l	Starten mit Assemblerdarstellung (der Optionsbuchstabe ist ein kleines L).
-p	Mausunterstützung.
-sc	Keine Unterscheidung zwischen Groß- und Kleinschreibung.
-sd*Verzeichnis* [*;Verzeichnis...*]	Verzeichnisse für Quellcodedateien.
-t*Verzeichnis*	Startverzeichnis zum Laden von Konfigurations- und ausführbaren Dateien.

B. Inline-Assembler-Befehle

In diesem Anhang sind die Befehlskurzformen und andere spezielle Symbole zur Eingabe von Befehlen mit dem Inline-Assembler aufgelistet. Es werden dieselben Kurzformen wie in Turbo Assembler und MASM verwendet.

Tabelle C.1 8086- / 80186- und 80286-Befehle

AAA	INC	LIDT**	REPNZ
AAD	INSB*	LLDT**	REPZ
AAM	INSW*	LMSW**	RET
AAS	INT	LOCK	RETF
ADC	INTO	LODSB	ROL
ADD	IRET	LODSW	ROR
AND	JB	LOOP	SAHF
ARPL**	JBE	LOOPNZ	SAR
BOUND*	JCXZ	LOOPZ	SBB
CALL	JE	LSL**	SCASB
CLC	JL	LTR**	SCASW
CLD	JLE	MOV	SGDT**
CLI	JMP	MOVSB	SHL
CLTS**	JNB	MOVSW	SHR
CMC	JNBE	MUL	SLDT**
CMP	JNE	NEG	SMSW**
CMPSB	JNLE	NOP	STC
CMPSW	JNO	NOT	STD
CWD	JNP	OR	STI
DAA	JO	OUT	STOSB
DAS	JP	OUTSB*	STOSW
DEC	JS	OUTSW*	STR**
DIV	LAHF	POP	SUB
ENTER*	LAR**	POPA*	TEST
ESC	LDS	POPF	WAIT
HLT	LEA	PUSH	VERR**
IDIV	LEAVE*	PUSHA*	VERW**
IMUL	LES	PUSHF	XCHG
IN	LGDT**	RCL	XLAT
			XOR

* nur ab 80186- und 80286-Prozessor verfügbar
** nur ab 80286-Prozessor verfügbar

Tabelle C.2 80386-Befehle

BSF	LSS	SETG	SETS
BSR	MOVSX	SETL	SHLD
BT	MOVZX	SETLE	SHRD
BTC	POPAD	SETNB	CMPSD
BTR	POPFD	SETNE	STOSD
BTS	PUSHAD	SETNL	LODSD
CDQ	PUSHFD	SETNO	MOVSD
CWDE	SETA	SETNP	SCASD
IRETD	SETB	SETNS	INSD
LFS	SETBE	SETO	OUTSD
LGS	SETE	SETP	JECXZ

Tabelle C.3 80486-Befehle

BSWAP	INVLPG
CMPXCHG	WBPINVD
INVD	XADD

Tabelle C.4 80386-/80486-Register

EAX	EDI
EBX	EBP
ECX	ESP
EDX	FS
ESI	GS

Tabelle C.5 CPU-Register

Byte-Register	ah, al, bh, bl, ch, cl, dh, dl
Wort-Register	ax, bx, cx, dx, si, di, sp, bp, Flags
Segmentregister	cs, ds, es, ss
Fließkommaregister	st, st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7)

Tabelle C.6 Reservierte Bezeichnungen

WORD PTR	TBYTE PTR
BYTE PTR	NEAR
DWORD PTR	FAR
QWORD PTR	SHORT

Tabelle C.7 8087- und 80287-Befehle

FABS	FIADD	FLDL2E	FST
FADD	FICOM	FLDL2T	FSTCW
FADDP	FICOMP	FLDPI	FSTENV
FBLD	FIDIV	FLDZ	FSTP
FBSTP	FIDIVR	FLD1	FSTSW*
FCHS	FILD	FMUL	FSUB
FCLEX	FIMUL	FMULP	FSUBP
FCOM	FINCSTP	FNOP	FSUBR

FCOMP	FINIT	FNSTSW**	FSUBRP
FCOMPP	FIST	FPATAN	FTST
FDECSTP	FISTP	FPREM	FWAIT
FDISI	FISUB	FPTAN	FXAM
FDIV	FISUBR	FRNDINT	FXCH
FDIVP	FLD	FRSTOR	FXTRACT
FDIVR	FLDCW	FSAVE	FYL2X
FDIVRP	FLDENV	FSCALE	FYL2XP1
FENI	FLDLG2	FSETPM*	F2XM1
FFREE	FLDLN2	FSQRT	

* nur ab dem 80287-Koprozessor verfügbar
** ab dem 80287-Koprozessor kann man bei den Befehlen `FSTSW` und `FNSTSW` auch das AX-Register zum Übertragen des Statusworts angeben

Tabelle C.8 80387-/80487-Befehle

FCOS	FUCOM
FSIN	FUCOMP
FPREM1	FUCOMPP
FSINCOS	

C. Die Konfiguration von Turbo Debugger

In diesem Kapitel erfahren Sie, wie Sie TDW mit Hilfe des Programms TDWINST für die Arbeit mit Windows-Anwendungen anpassen können. Dieses Kapitel umfaßt folgende Schwerpunkte:

1. Der Aufruf von TDWINST
2. Einstellung der Bildschirmfarben
3. Die Einstellung der Bildschirmparameter in TDW
4. Die Einstellung der TDW-Optionen
5. Das Einstellen des Bildschirmmodus
6. Kommandozeilenoptionen und Installationsoptionen
7. Wenn alles wie gewünscht eingestellt wurde

TDW ist startbereit, sobald Sie von den ausgelieferten Disketten Arbeitskopien erstellt haben. Mit Hilfe des Installationsprogramms TDWINST.EXE haben Sie allerdings die Möglichkeit, eine Anzahl von Standardoptionen zu verändern. Auch durch Angabe von Optionen in der Kommandozeile beim Start von TDW können Sie einige Einstellungen des Debuggers Ihrer Verwendung gemäß ändern. Sollten Sie dabei feststellen, daß Sie manche Kommandozeilenoptionen immer wieder benötigen, können Sie diese mit Hilfe des Installationsprogramms fest abspeichern. Folgende Einstellungen lassen sich mit dem Installationsprogramm verändern:

- Farben der Fenster, Dialogfenster und Menüs.

- Bildschirmparameter: Auslagerung des Bildschirms, Integerdarstellung, Startbildschirm (Quell- oder Assemblercode), Anzahl der Bildschirmzeilen, Tabulatoreinstellung, maximale Größe nicht-überlappender *Watches* -Fenster, Flimmerkontrolle, 43-/50-Zeilenmodus, Sichern des Grafikspeichers, Art der Bildschirmaktualisierung, Anzahl der Nachkommastellen bei Gleitkommazahlen, Anzahl der darzustellenden Zeigerelemente und Umfang des Protokolls.

- Suchpfade für die Quellcodedateien und die TDW-Hilfs- und Konfigurationsdateien.

- Eingabe- und Tastaturparameter: Interrupt-Taste zur Programmunterbrechung, Länge der History-Liste, akustische Warnung bei Fehlern, Mausunterstützung, Aufzeichnung von Tastaturbefehlen sowie Strg -Tastenkürzel.
- Debuggen im Quellcode: Optionen der Programmiersprachen.
- Bildschirmmodus

Der Aufruf von TDWINST

Sie starten das Installationsprogramm durch Eingabe von TDWINST auf der DOS-Ebene oder vom Windows-Program-Manager aus mit dem Befehl *File/Run*. Sobald TDWINST startet, erscheint das dazugehörige Hauptmenü. Anschließend können Sie entweder einen der unterlegten Anfangsbuchstaben der Menüoptionen eingeben oder mit Hilfe der beiden Pfeiltasten, ↑ und ↓, den gewünschten Menüpunkt ansteuern und dann die Eingabetaste CR drücken. Mit der Eingabe von D können Sie zum Beispiel die Bildschirmdarstellung ändern. Die anderen Menüs des Installationsprogramms werden ebenso bedient. Um zum übergeordneten Menü zurückzukommen, drücken Sie die Taste Esc und wiederholen dies gegebenenfalls, bis das Hauptmenü wieder aktiv ist.

Einstellung der Bildschirmfarben

Durch die Auswahl von *Colors* erhalten Sie das *Colors*-Menü. Nun haben Sie die Möglichkeit, zwischen zwei Menüpunkten zu wählen. *Customize* (engl. individuell gestalten) und *Default Color Set* (engl. Standardfarbpalette).

Die Änderung der Bildschirmfarben

Wählen Sie *Customize* aus, erscheint ein drittes Menü mit Optionen für die individuelle Gestaltung von Fenstern, Menüs und Bildschirmen.

Windows

Für die Anpassung von Fenstern wählen Sie den Befehl *Windows*. Dadurch wird ein viertes Fenster geöffnet, in dem Sie den Fenstertyp auswählen können, der verändert werden soll: *Text*, *Data*, *Low Level* (in diesem Fall das *CPU*-Fenster) und *Other* (zum Beispiel das *Breakpoints*-Fenster). Durch die Auswahl einer dieser Optionen erhalten Sie ein weiteres Fenster mit den Fensterelementen und einer Darstellung von Beispielfenstern (ein aktives und ein nicht aktives), mit der Sie

die verschiedenen Farbkombinationen testen können. Der Bildschirm sieht wie folgt aus:

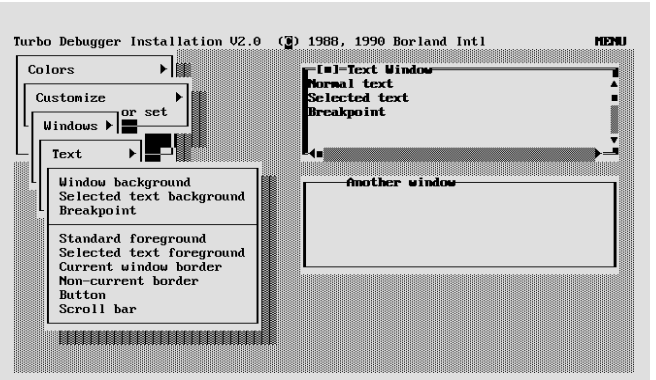


Abbildung C.1 Änderung der Fensterfarben

Wenn Sie ausgewählt haben, was Sie verändern möchten, erscheint ein Kästchen mit der Farbpalette über dem Menü. Mit Hilfe der Pfeiltasten können Sie sich innerhalb des Palettenkästchens bewegen. Rechts davon sehen Sie alle Fenster oder Kästchen in der Kategorie, die Sie verändern. Wenn Sie nun den Cursor durch die Farbauswahl bewegen, so erscheint das Fenster, das Sie verändern wollen, in der jeweiligen Farbe, damit Sie sehen können, ob Ihnen die Farbeinstellung zusagt. Wenn Sie die richtige Einstellung gefunden haben, drücken Sie die Eingabetaste [CR], um den Vorgang abzuschließen.

Achtung: TDW verwaltet vier Farbtabellen: für Farb-, Schwarzweiß-, Monochrom- und LCD-Bildschirme. Sie können davon nicht mehrere gleichzeitig verändern, sondern immer nur die Tabelle, die von Ihrem aktuellen Bildschirmmodus und der Art Ihres Bildschirms abhängt. Wenn Sie also mit einem Farbbildschirm arbeiten und die Schwarzweißtabelle verändern möchten, müssen Sie zuerst Ihren Bildschirmmodus auf der DOS-Ebene (`mode BW80`) ändern und dann TDWINST erneut starten. Um auf farbige Darstellung zurückzuschalten, geben Sie `mode CO80` ein.

Dialogs

Wenn Sie im Menü *Customize* den Menüpunkt *Dialogs* ausgewählt haben, erscheint eine Liste mit Dialogfenstern und Menüpunkten sowie ein Dialogfenster, mit dem Sie experimentieren können, dargestellt in Abbildung C.2 auf der folgenden Seite.

Wie im *Windows*-Menü wird durch die Auswahl eines Menüpunkts eine Palette geöffnet, aus der Sie die gewünschte Farbe für dieses Element auswählen können.

Menus

Nach Auswahl von *Menus* erscheint ein Menü mit den verfügbaren Menüpunkten zusammen mit einem Testmenü. Wählt man ein Element in diesem Menü aus, so erscheint die gewohnte Farbpalette.

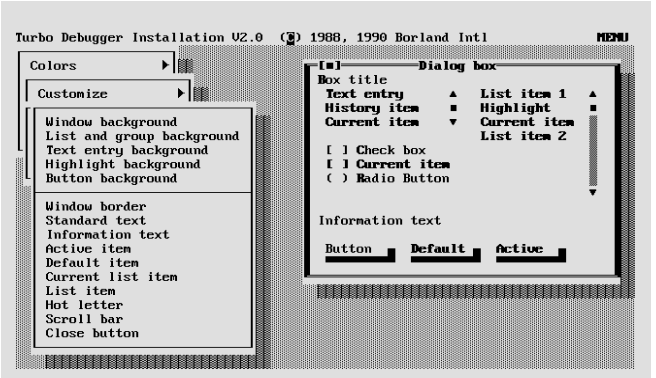


Abbildung C.2 Anpassung der Farben für ein Dialogfenster

Screen

Durch die Auswahl von *Screen* erscheint ein Menü, aus dem Sie auf ein weiteres Menü zugreifen können, in dem Bildschirmmuster und Farbpaletten für die einzelnen Bildschirmelemente enthalten sind. Außerdem erscheint ein Testhintergrund, mit dem Sie die Auswahl überprüfen können.

Die Standardfarben

Wenn Sie im *Colors*-Menü den Punkt *Default Color Set* auswählen, können Sie die Farbpaletten für Text-, Daten- und sonstige Fenster überprüfen; für jede Fensterart erscheint ein aktives Fenster und ein nicht aktives Fenster auf dem Bildschirm, in denen Sie die Standardfarben für die einzelnen Elemente überprüfen können.

Die Einstellung der Bildschirmparameter in TDW

Wenn Sie im Hauptmenü die Option *Display* auswählen, erhalten Sie das Dialogfenster *Display options* .

Mit Hilfe der Tabulatortaste können Sie zwischen den verschiedenen Optionen hin- und herschalten. Zum Umschalten der Auswahlfelder dienen die Tasten ↑ und ↓; zum Auswählen eines Markierungsfeldes verwenden Sie die Leertaste.

Hinweis: Einige der dargestellten Optionen können Sie sowohl mit TDWINST fest einstellen als auch in der Kommandozeile von TDW angeben. Am Ende dieses Anhangs finden Sie eine Übersicht über alle Kommandozeilenoptionen und die entsprechenden Einstellungen in TDWINST.

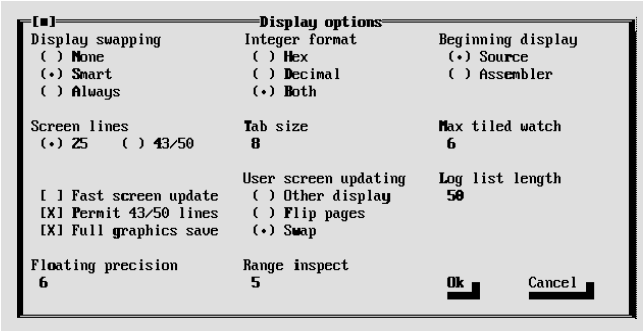


Abbildung C.3 Das Dialogfenster Dialog options

Display swapping

Mit den Auswahlfeldern unter *Display swapping* legen Sie fest, wie TDW zwischen seinem eigenen Bildschirm und dem Programmbildschirm umschaltet. Mit den Tasten ↑ und ↓ können Sie zwischen folgenden Einstellungen wählen:

Smart Schaltet Sie nur dann auf den Benutzerbildschirm um, wenn eventuell etwas ausgegeben wird. TDW schaltet jedesmal um, wenn Sie bei der Einzelschrittausführung zu einer Routine kommen oder eine Zeile ausführen, in der etwas auf dem Bildschirm ausgegeben wird. Das ist die Standardeinstellung.

Always Schaltet jedesmal um, wenn das Benutzerprogramm läuft. Verwenden Sie diese Option, wenn TDW im Smart-Modus nicht alle Ausgaben erwischt. Wenn Sie diese Option verwenden, flackert der Bildschirm im Einzelschrittverfahren bei jedem Schritt auf, da der Bildschirm von TDW mit dem Programmbildschirm ausgetauscht werden muß.

Integer format

Mit den Auswahlfeldern unter *Integer format* können Sie bestimmen, wie Integer-Zahlen dargestellt werden sollen. Sie können folgende Einstellungen vornehmen:

Hex für eine Darstellung als Hexadezimalzahlen.

Decimal für eine Darstellung als Dezimalzahlen.

Both für eine Darstellung beider Systeme.

Beginning Display

Mit den Auswahlfeldern unter *Beginning display* legen Sie fest, in welcher Sprache Ihr Programm nach dem Aufruf dargestellt wird. Zwei Einstellungen sind möglich:

Source	Quellcodestart: Der Initialisierungscode Ihres Compilers wird abgearbeitet. Sie beginnen Ihre Arbeit in einem *Module*-Fenster, in dem das Hauptprogramm dargestellt ist.
Assembler	Assemblerstart: Der Startup-Code Ihres Programme wird nicht ausgeführt. Ein *CPU*-Fenster zeigt den ersten Befehl in Ihrem Programm an.

Screen lines

Mit Hilfe dieser Auswahlfelder schalten Sie zwischen dem 25- und dem 43/50-Zeilenmodus um.

Achtung: Nur mit EGA-, VGA- oder SVGA-Ausrüstung lassen sich mehr als 25 Zeilen darstellen.

Tab size

In diesem Eingabefeld wird die Spaltenbreite der Tabulatoren in Text- oder Dateifenstern festgelegt. Sie werden nach der Anzahl der Spalten gefragt (die Anzahl muß zwischen 1 und 32 liegen); die Vorgabe ist 8.

Maximum tiled watch

In diesem Eingabefeld geben Sie an, auf wieviele Zeilen ein *Watches* -Fenster im nicht überlappenden Modus maximal ausgedehnt werden kann. Zulässig ist eine Eingabe von 1 bis 20 Zeilen; die Voreinstellung liegt bei 6 Zeilen.

Fast screen update

Im Markierungsfeld *Fast screen update* können Sie Ihr System auf »schnelle« Bildschirmausgabe einstellen. Schalten Sie diese Option aus, wenn es bei der schnellen Ausgabe zu Bildschirmstörungen (»Schnee«) kommt. Wenn Sie der »Schnee« nicht stört, brauchen Sie diese Option nicht auszuschalten; es gibt

Anwender, denen ein schneller Bildschirmaufbau wichtiger ist als ein flimmerfreies Bild.

Permit 43/50 line mode

In diesem Markierungsfeld legen Sie fest, ob der 43-Zeilen-Modus oder der 50-Zeilen-Modus zugelassen wird. Falls Sie diese Option ausschließen, können Sie etwa 8 KB an Speicherplatz einsparen, da ein Bildschirmmodus mit mehr Zeilen in TDW einen größeren Speicherplatz für die Fenster erfordert. Bei sehr umfangreichen Programmen, die auf ein Maximum an Speicherplatz angewiesen sind, kann sich das Abschalten diese Option positiv auswirken. Wenn Sie diese Option abgeschaltet haben, können Sie den Bildschirm nicht mehr in den 43/50-Zeilenmodus umschalten, auch wenn Ihr System dazu in der Lage wäre.

International sort

Dieses Markierungsfeld dient zur Aktivierung der landesspezifischen Sortierreihenfolge; es gilt die alphabetische Reihenfolge der Sprache, die im Dialogfenster *International* des *Windows Control Panel* angezeigt wird. Wenn Sie *International Sort* nicht aktivieren, werden die Einträge in Feldern, die Auflistungen enthalten, entsprechend ihrer ANSI-Werte sortiert.

User screen updating

Mit den Auswahlfeldern unter *User screen updating* legen Sie fest, auf welche Weise der Bildschirm aktualisiert wird, wenn zwischen der Darstellung Ihres Programms und der Darstellung von TDW hin- und hergeschaltet wird. Mit *User screen updating* können Sie zwischen folgenden Optionen umschalten:

Other display TDW wird auf einem zweiten angeschlossenen Bildschirm dargestellt. Wenn Sie sowohl einen Farb- als auch einen Monochrombildschirmadapter haben, können Sie mit dieser Option Ihren Programmbildschirm auf dem einen Bildschirm und TDW auf dem anderen gleichzeitig darstellen.

Swap Benützt einen einzigen Bildschirmadapter und eine Bildschirmoberfläche. Die Inhalte des Benutzerbildschirms und des Turbo-Debugger-Bildschirms werden durch die Software zwischen Hauptspeicher und Bildschirmadapter hin- und hergeladen. Dies ist die langsamste Methode des Bildschirmwechsels, aber auch die sicherste und diejenige, die das Benutzerprogramm am geringsten beeinflußt. Wählen Sie diese Option,

wenn Sie ein Programm mit mehreren logischen Bildschirmseiten, z.B. ein Grafikprogramm debuggen.

Log list length

In diesem Eingabefeld legen Sie fest, wieviele Ereignisse im Protokoll zwischengespeichert werden sollen. Vorgegeben ist der Wert 50, zulässig sind Werte zwischen 4 und 200.

Floating precision

In diesem Eingabefeld wird die Anzahl der von TDW zu berücksichtigenden Nachkommastellen von Gleitkommazahlen festgelegt. Der Wert ist standardmäßig auf 6 festgelegt; es können Werte zwischen 1 und 32 eingegeben werden.

Range Inspect

In diesem Eingabefenster wird die Anzahl der Zeigerelemente festgelegt, die TDW darstellt, nachdem Sie in einem Inspector-Fenster das *Range*-Kommando des lokalen Menüs gewählt haben. Standardmäßig gilt hier der Wert 5, was bedeutet, daß in TDW nur die ersten fünf zu diesem Zeiger gehörenden Elemente dargestellt werden, sofern Sie nicht im *Range*-Kommando des lokalen Menüs eine andere Angabe machen. Der größte zulässige Wert beträgt 4096.

Wenn irgend eines der Eingabefelder aus dem Dialogfenster Display Options leer sind wenn Sie versuchen, es zu verlassen, werden Sie aufgefordert, einen Wert einzugeben, bevor Sie es verlassen.

Die Einstellung der TDW-Optionen

Durch die Auswahl des Befehl *Options* erhalten Sie ein Menü mit Optionen, die wiederum zu Dialogfenstern führen.

Directories

In diesem Dialogfenster befinden sich Eingabefelder, in denen folgende Aufrufbefehle und Verzeichnispfade angegeben werden können:

Source directories gibt die Verzeichnisse an, in denen TDW nach Quelldateien sucht.

Turbo directory enthält das Verzeichnis, in dem TDW nach der Hilfedatei und der Konfigurationsdatei suchen soll.

Input & Prompting

In diesem Dialogfenster können Sie einstellen, wie Eingaben in TDW verarbeitet und entgegengenommen werden sollen. Hier können Sie auch festlegen, ob TDW ein vollständiges Protokoll Ihrer Debug-Sitzung anfertigen soll.

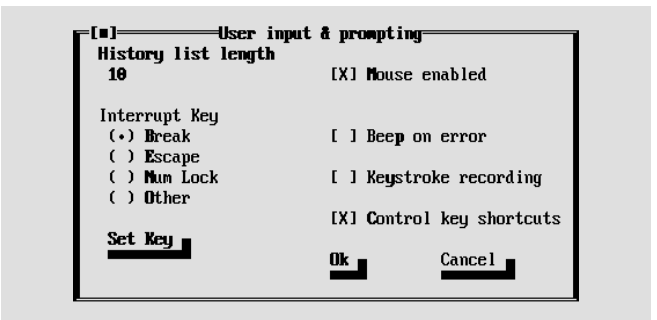


Abbildung C.4 Das Dialogfenster User Input & Prompting

History list length

In diesem Eingabefeld können Sie angeben, wieviele Einträge in einer History-Liste erhalten bleiben sollen. Per Voreinstellung bleiben 10 Einträge erhalten; es können hier Werte zwischen 1 und 40 eingegeben werden.

Beep on error

Standardmäßig gibt TDW einen Alarmton aus, wenn Sie eine unzulässige Taste drücken oder eine Fehlermeldung verursachen. Mit dem Markierungsfeld *Beep on error* können Sie diese Voreinstellung verändern.

Control key shortcut

Dieses Markierungsfeld schaltet die Strg -Tastenkürzel (engl. Ctrl) ein oder aus. Wenn die Strg -Tastenkürzel eingeschaltet sind, kann man jeden Befehl in einem lokalen Menü direkt mit der Strg -Taste und dem ersten Buchstaben des Menüpunkts aufrufen. Man kann dann aber die WordStar-ähnlichen Befehle für die Cursorbewegungen nicht verwenden.

Mouse enabled

Dieses Markierungsfeld legt fest, ob TDW standardmäßig eine Maus unterstützt.

Full trace history

Diese Option können Sie auswählen, wenn TDW eine vollständige History-Liste ihrer Debug-Sitzung anfertigen soll. Mit Hilfe einer History-Liste können Sie sich rückwärts durch den Quellcode Ihres Programms durcharbeiten; wenn diese Option aktiv ist, verlangsamt sich allerdings die Ausführungsgeschwindigkeit Ihres Programms.

Source Debugging

Im Dialogfenster *Source Debugging* legen Sie fest, welche Programmiersprache für die Auswertung von Ausdrücken verwendet und ob zwischen Groß- und Kleinschreibung unterschieden werden soll; außerdem wird hier festgelegt, ob Breakpoints für Windows-Meldungen in ObjectWindows-Programmen unterstützt werden oder nicht.

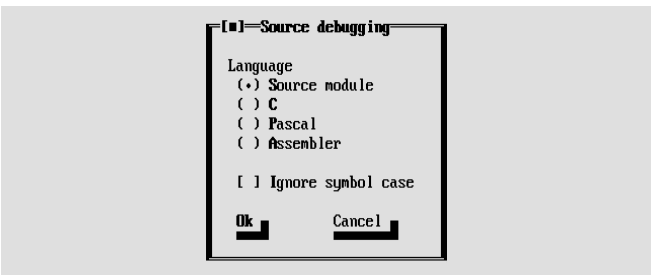


Abbildung C.5 Das Dialogfenster Source Debugging

Language

In den Auswahlfeldern von *Language* kann festgelegt werden, in welcher Sprache TDW Ausdrücke auswerten soll. Sie haben die Auswahl zwischen:

Source module	Die Programmiersprache wird anhand der Sprache des aktuellen Quellcodes ausgewählt.
C	Verwendet ausschließlich C-Ausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.
Pascal	Verwendet ausschließlich Pascal-Ausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.
Assembler	Verwendet ausschließlich Assemblerausdrücke, egal in welcher Sprache die aktuellen Module geschrieben sind.

Ignore symbol case

Wenn dieses Feld markiert ist, wird zwischen Groß- und Kleinschreibung nicht unterschieden. Dies kann man ändern, indem man die Markierung in diesem Feld aufhebt.

OWL window messages
Diese Option ist standardmäßig nicht aktiviert; in diesem Fall gibt es beim Debuggen von ObjectWindows-Programmen keine Unterstützung für Windows-Botschaften, und TDW erkennt keine Windows-Objekte und Windows-Prozeduren. Wenn Sie trotzdem Breakpoints für Windows-Botschaften setzen wollen, müssen Sie Windows-Handles verwenden (und außerdem ausreichende Kenntnisse über ObjectWindows besitzen, um ein Handle lokalisieren zu können).

Wenn Sie dieses Markierungsfeld aktivieren, unterstützt TDW Breakpoints für Windows-Botschaften in ObjectWindows-Programmen auf eine besondere Weise. Es ist dann möglich, beim Setzen eines Breakpoints für eine Windows-Botschaft den Namen eines Fensterobjekts aus dem Programm zu verwenden. Wenn diese Option aktiv ist, wird der Ablauf allerdings langsamer.

Das Einstellen des Bildschirmmodus

Unter diesem Menüpunkt legen Sie den Bildschirmmodus fest:

Default
TDW stellt die Art Ihres Bildschirmadapters fest und wählt den Modus aus, der hierfür am besten geeignet ist.

Color
Sie erhalten eine farbige Darstellung, wenn Sie diese Einstellung als Vorgabeeinstellung setzen und einen der folgenden Grafikadapter verwenden: CGA, MCGA, EGA, VGA, SVGA oder 8514 von IBM.

Black and White
Sie erhalten Schwarzweiß-Darstellung, wenn Sie diese Einstellung als Vorgabeeinstellung setzen und Sie einen der folgenden Grafikadapter verwenden: CGA, MCGA, EGA, VGA, SVGA oder 8514 von IBM.

Monochrome
Wählen Sie diese Einstellung, wenn Sie einen Farbmonitor zusammen mit einer Herculeskarte oder einer Monochrome-Karte verwenden.

LCD
Wählen Sie diese Einstellung, wenn Sie einen LCD-Monitor verwenden.

Kommandozeilenoptionen und Installationsoptionen

Einige der Optionen, die im vorliegenden Abschnitt beschrieben wurden, kann man auch als Kommandozeilenoptionen beim Aufruf von TDW angeben. Die folgende Tabelle zeigt Entsprechungen zwischen den Kommandozeilenoptionen von TDW und den Befehlen des Installationsprogramms TDWINST, mit dem man die Optionen dauerhaft abspeichern kann.

Tabelle D.1 Kommandozeilenoptionen von TDW

Option	TDWINST-Menüauswahl	Dialogfenster
	Display	*Display Options*
	Display	*Display Options*
-do		(•) Other Display
-ds		(•) Swap
	Display	*Display Options*
-l		(•) Assembler
-l-		(•) Source
	Options / Input & Prompting	*User Input & Prompting*
-p		[X] Mouse enabled
-p-		[ ] Mouse enabled
	Options / Directories	*Directories*
-sd		Source Directories
-t		Startverzeichnis

Wenn Sie beispielsweise TDW von der DOS-Ebene aus starten, geben Sie ein:

```
win TDW -p
```

Dieses Kommdando lädt Windows und ermöglicht die Verwendung einer Maus. Vergessen Sie nicht, daß Sie win eingeben müssen, wenn Sie TDW von der DOS-Ebene aus starten, da für den Start von TDW Windows von Microsoft benötigt wird. Denken Sie auch daran, daß TDW.EXE sich im aktuellen Verzeichnis befinden muß (oder in einem Verzeichnis, das im Pfadeintrag aufgeführt ist), damit dieses Kommando angwendet werden kann.

Eine Liste aller Kommandozeilenoptionen für TDW.EXE und TDWINST.EXE erhalten Sie, wenn Sie den Programmnamen eingeben und anschließend -h oder -?.

Wenn alles wie gewünscht eingestellt wurde

Speichern der Änderungen

Nachdem Sie alle Optionen von TDW wie gewünscht eingestellt haben, wählen Sie den Befehl *Save* des Hauptmenüs aus, um festzulegen, wie die Konfiguration gespeichert werden soll.

Save Configuration File

Wenn Sie *Save Configuration File* wählen, wird ein Dialogfenster geöffnet, dessen Eingabefeld mit dem Standarddateinamen TDCONFIG.TDW initialisiert wird. Starten Sie TDWINST mit der Kommandozeilenoption *-c*, erscheint hier der Dateiname, den Sie nach dieser Option angegeben haben.

Sie können diesen Namen mit CR übernehmen oder einen anderen Namen für die Konfigurationsdatei eingeben. Falls Sie einen anderen Dateinamen angeben, können Sie die Datei beim Start von TDW mit der Kommandozeilenoption *-c* laden. Zum Beispiel:

```
tdw -cspezial.kfg prox
```

Die Datei kann aber auch mit dem Befehl *Options/Restore configuration* während der Arbeit mit TDW eingelesen werden.

Es gibt also mehrere Möglichkeiten, TDW mit Hilfe der Konfigurationsdatei MYCFG.TDW zu starten. Wenn Windows bereits geladen ist, geben Sie von der Befehlszeile des Kommandos *File / Run* aus dem Windows-Program-Manager die folgende Zeile ein (die Konfigurationsdatei MYCFG.TDW muß sich in einem Verzeichnis befinden, für das ein Pfad-Eintrag existert):

```
tdw -cmycfg.tdw myprog
```

Sie können auch den TDW-Befehl *Options / Restore Options* verwenden, um eine Konfiguration zu laden, nachdem Sie TDW bereits gestartet haben.

Modify TDW.EXE

Wenn Sie *Modify TDW.EXE* wählen, werden alle Konfigurationsänderungen direkt in der Programmdatei von TDW, TDW.EXE, gespeichert. Beim nächsten Aufruf von TDW werden diese Einstellungen standardmäßig vorgegeben.

Achtung: Wenn Sie zu irgendeinem Zeitpunkt wieder auf die Standard-Konfiguration von TDW zurückgreifen wollen, kopieren Sie TDW.EXE von Ihren Original-Disketten auf Ihre Arbeitsdiskette und überschreiben die veränderte TDW.EXE-Datei.

TDWINST verlassen

Mit dem Befehl *Quit* des Hauptmenüs können Sie TDWINST jederzeit verlassen.

D. Dialogfenster und Fehlermeldungen

TDW gibt Fehlermeldungen und Dialogfenster an der aktuellen Cursor-Position aus. In diesem Kapitel werden alle Dialogfenster und Fehlermeldungen beschrieben, die von TDW erzeugt werden.

Hier wird beschrieben, wie Sie auf Dialogfenster oder Fehlermeldungen reagieren können. Alle Dialogfenster und Fehlermeldungen (inklusive der schwerwiegenden Fehler beim Programmstart) sind in alphabetischer Reihenfolge aufgeführt und werden einzeln beschrieben.

Meldungen in Dialogfenstern

TDW öffnet ein Dialogfenster, wenn zusätzliche Informationen benötigt werden, um einen Befehl abzuschließen. Die Überschrift des Dialogfensters beschreibt kurz, welche Informationen eingegeben werden sollen. In manchen Dialogfenstern steht eine History-Liste mit bereits früher eingegebenen Antworten zur Verfügung.

Es gibt zwei Möglichkeiten, ein Dialogfenster zu bearbeiten:

- Sie geben die Antwort ein und drücken CR
- Sie drücken Esc, um den Dialog abzubrechen und zu dem Menübefehl zurückzukehren, mit dem das Dialogfenster aufgerufen wurde.

In manchen Dialogfenstern haben Sie genau zwei Antwortmöglichkeiten (wie *Yes/No*). Mit Hilfe der Tab -Taste können Sie eine der beiden Möglichkeiten wählen. Anschließend bestätigen Sie Ihre Auswahl mit CR oder wählen direkt Y oder N. Mit Hilfe von Esc kann die Ausführung des Befehls sofort beendet werden.

Eine vollständige Besprechung der Tastaturbefehle in Dialogfenstern finden sie in Kapitel 2, »Fehlersuche mit Turbo Debugger«.

Hier nun eine alphabetische Auflistung aller Meldungen, die von Dialogfenstern erzeugt werden:

Already recording, do you want to abort?
[Aufnahme läuft bereits, wollen Sie abbrechen?]
Sie nehmen bereits ein Tastaturmakro auf. Sie können nicht mit der Aufnahme eines neuen Tastaturmakros beginnen, bevor die laufende Aufnahme beendet ist. Geben Sie Y ein, um die Aufnahme des Makros zu beenden, N um die Aufnahme fortzusetzen.

Device error – Retry?
[Gerätefehler – Noch einmal probieren?]
Beim Schreiben auf ein Zeichenausgabegerät, zum Beispiel auf den Drucker, ist ein Fehler aufgetreten. Die Ursache kann entweder darin liegen, daß der Drucker nicht angeschlossen ist oder kein Papier mehr hat. Korrigieren Sie den Sachverhalt und geben Sie dann Y ein, um es nochmal zu versuchen, oder brechen Sie den Vorgang durch Eingabe von N ab.

Disk error on drive ___ – Retry?
[Laufwerksfehler auf Laufwerk ___ – Noch einmal probieren?]
Beim Zugriff auf das angegebene Laufwerk ist ein schwerwiegender Fehler aufgetreten. Das kann heißen, daß Sie entweder keine Diskette im Laufwerk haben, oder, falls eine Festplatte vorhanden ist, daß die Festplatte einen nicht mehr lesbaren oder beschreibbaren Sektor aufweist. Sie können Y eingeben, um zu sehen, ob ein zweiter Versuch zum Erfolg führt, sonst geben Sie N ein, um den Vorgang abzubrechen.

Edit watch expression
[Watch-Ausdruck editieren]
Sie müssen den Ausdruck im *Watches* -Fenster ändern oder ersetzen. Das Dialogfenster wird mit dem hervorgehobenen Watch-Ausdruck initialisiert.

Enter address, count, byte value
[Adresse, Anzahl und Initialisierungswert eingeben]
Geben Sie die Adresse des Speicherblocks ein, den Sie mit einem bestimmten Wert initialisieren möchten. Anschließend geben Sie die Anzahl der Bytes ein, die Sie initialisieren wollen. Als drittes Argument muß der Wert angegeben werden, mit dem Sie den Speicherbereich füllen wollen.

Enter adress to position to
[Geben Sie die Adresse ein, zu der Sie springen wollen]
Geben Sie die Adresse ein, die Sie in Ihrem Programm sehen wollen. Sie können einen Funktionsnamen, eine Zeilennummer, eine absolute Adresse oder einen Zeigerausdruck eingeben. In Kapitel 9, »Ausdrücke«, finden Sie weitere Informationen über die Eingabemöglichkeiten von Adressen.

Enter animate delay (10ths of sec)
[Geben Sie die Zeitspanne für die Verzögerung (in Zehntelsekunden) in der Animation ein]
Geben Sie an, wie schnell der Befehl *Animate* ausgeführt werden soll. Je höher

die eingegebene Zahl ist, desto länger ist die Zeitspanne zwischen zwei aufeinanderfolgenden Schritten während der Animation.

Enter code address to execute to
[Geben Sie die Codeadresse ein, bis zu der das Programm laufen soll]
Geben Sie die Adresse in Ihrem Programm ein, an der das Programm unterbrochen werden soll. In Kapitel 9, »Ausdrücke«, finden Sie eine Erläuterung über die Möglichkeiten der Adreßeingabe.

Enter command-line arguments
[Geben Sie die Kommandozeilenargumente ein]
Geben Sie die Kommandozeilenargumente für das Programm ein, das Sie debuggen wollen.

Enter comment to add to end of log
[Geben Sie einen Kommentar ein, der am Ende des Protokolls angefügt werden soll]
Geben Sie eine beliebige Textzeile ein, die im *Log*-Fenster angehängt werden soll. Die Zeile wird dem Protokoll genau wie eingegeben angefügt.

Enter destination adress for marked block
[Geben Sie die Zieladresse für den markierten Block ein]
Geben Sie in der Segment- oder in der Segment:Offset-Schreibweise die Adressen ein, zu der Sie den Block verlagern wollen.

Enter expression for conditional breakpoint
[Geben Sie einen Ausdruck für einen bedingten Breakpoint ein]
Geben Sie einen Ausdruck ein, der zutreffen muß (TRUE), wenn der Breakpoint ausgelöst werden soll. Der Ausdruck wird jedesmal ausgewertet, wenn das Programm zum Breakpoint kommt. Achten Sie auf eventuelle Seiteneffekte.

Enter expression to watch
[Geben Sie den Ausdruck ein, der überwacht werden soll]
Geben Sie einen Variablennamen oder einen Ausdruck ein, dessen Wert Sie im *Watches* -Fenster überwachen wollen. Wenn Sie wollen, dann können Sie auch einen Ausdruck eingeben, der sich nicht auf eine Speicherstelle bezieht, wie zum Beispiel `x * y + 4`. Wenn das Dialogfenster mit einem Textausschnitt initialisiert wurde, kann man den Eintrag mit CR sofort akzeptieren oder verändern und etwas anderes eingeben.

Enter inspect start index, range
[Geben Sie den Index des ersten Elements und den Bereich ein]
Geben Sie den Index des ersten Array-Elements ein, das Sie betrachten wollen. Als zweites Argument müssen Sie die Anzahl der Elemente eingeben. Die beiden Zahlen müssen mit einem Leerzeichen oder einem Komma (,) voneinander getrennt werden.

Enter instruction to assemble
[Geben Sie den Befehl ein, der assembliert werden soll]
Geben Sie eine Assembleranweisung ein, die an der aktuellen Adresse des Code-Ausschnitts eingefügt werden soll. In Anhang C finden Sie eine Aufstellung mit allen zulässigen Assemblerbefehlen. In Kapitel 11, »Debuggen auf Assembler-Ebene«, wird der Einsatz von Assemblerbefehlen ausführlich behandelt.

Enter log file name
[Geben Sie den Namen der Protokolldatei an]
Geben Sie den Namen der Datei an, in die das Protokoll des *Log*-Fensters geschrieben werden soll. Bis zu dem Zeitpunkt, da Sie den Befehl *Close log file* aufrufen, werden alle Zeilen, die im *Log*-Fenster ausgegeben werden, auch in die Datei geschrieben. Die Voreinstellung für den Dateinamen ist die Dateierweiterung ».LOG« mit dem Namen der gerade bearbeiteten Datei. Mit CR können Sie diese Vorgabe akzeptieren oder Sie geben stattdessen einen neuen Namen ein.

Enter memory address, count
[Geben Sie die Adresse und Länge eines Speicherbereichs ein]
Geben Sie eine Speicherplatzadresse ein. Optional kann nach einem Komma die Anzahl der Elemente angegeben werden. Sie können einen Symbolnamen oder einen vollständigen Ausdruck verwenden.

Enter name of file to view
[Geben Sie den Namen der Datei ein, die dargestellt werden soll]
Sie können entweder ein Jokerzeichen, wie es von DOS verwendet wird, eingeben, um eine Liste mit Dateien zu erhalten, oder Sie geben den Dateinamen direkt ein.

Enter new bytes
[Geben Sie die neuen Bytes ein]
Geben Sie eine Liste von Bytes ein, um die Bytes zu ersetzen, die an der Stelle der Datei stehen, an der sich der Cursor befindet. In Kapitel 9, »Ausdrücke«, finden Sie eine vollständige Beschreibung von Byte-Folgen.

Enter new coprocessor register value
[Geben Sie den neuen Wert für das Register des Koprozessors ein]
Geben Sie einen neuen Wert für das Register des Arithmetikprozessors ein, das momentan hervorgehoben ist. Sie können einen vollständigen Ausdruck eingeben, um den neuen Wert zu erzeugen. Der Ausdruck wird auf das richtige Fließkommaformat konvertiert, bevor er in das Register geladen wird.

Enter new data bytes
[Geben Sie die neuen Daten-Bytes ein]
Geben Sie eine Liste von Bytes ein, um die Bytes zu ersetzen, die an der Stelle des Segments stehen, das vom Cursor gekennzeichnet ist. In Kapitel 9, »Ausdrücke«, finden Sie eine vollständige Beschreibung der Byte-Folgen.

Enter new directory
[Geben Sie ein neues Verzeichnis ein]
Geben Sie die neue Laufwerksbezeichnung und/oder den Namen des Verzeichnisses ein, das Sie als aktuelles Laufwerk oder Verzeichnis verwenden möchten.

Enter new file offset
[Geben Sie neuen Datei-Offset ein]
Sie untersuchen eine auf Platte befindliche Datei in hexadezimaler Schreibweise; geben Sie den Offset des Anfangs der Datei ein, deren Datenbytes Sie ansehen wollen. Die Datei wird dann auf die Zeile positioniert, die den von Ihnen angegebenen Offset besitzt.

Enter new line number
[Geben Sie die Nummer der neuen Zeile ein]
Geben Sie die Nummer der Zeile des aktuellen Moduls ein, die Sie sehen wollen. Wenn Sie eine Zeilennummer eingeben, die hinter dem Ende der Datei liegt, kommen Sie in die letzte Zeile. Die Numerierung der Zeilen beginnt mit Eins. Die aktuelle Zeile wird durch den Cursor im *Module*-Fenster angezeigt.

Enter new relocation segment value
[Geben Sie einen neuen Wert für das Basissegment ein]
Geben Sie einen Ausdruck in der aktuellen Programmiersprache ein. Dieser Wert legt die Adresse des Basissegments der Symboltabelle fest, die mit dem Befehl *File/Symbol Load* geladen wurde. Der eingegebene Ausdruck sollte dem Segment entsprechen, an dem der Programmcode der entsprechenden Symboltabelle beginnt.

Enter new selector
[Geben Sie einen neuen Selektor ein]
Geben Sie den Selektor-Wert ein, der von nun an der aktuelle sein soll. Sie können den Wert in hexadezimaler Form eingeben oder als Segment, also etwa CS, DS oder ES.

Enter new value
[Geben Sie einen neuen Wert ein]
Geben Sie für das momentan hervorgehobene CPU-Register einen neuen Wert ein. Sie können einen vollständigen Ausdruck eingeben, um den neuen Wert festzulegen.

Enter port number
[Geben Sie die Portadresse ein]
Geben Sie die Adresse des Eingabeports an, den Sie lesen wollen. Als Portadresse sind Werte zwischen 0 und 65535 zulässig.

Enter port number, value to output
[Geben Sie die Portadresse und den Wert ein, der ausgegeben werden soll]
Geben Sie die Adresse des Ausgabeports und den Wert an, der ausgegeben

werden soll. Die beiden Ausdrücke müssen durch ein Komma voneinander getrennt werden. Als Portadresse sind Werte zwischen 0 und 65535 zulässig.

Enter program name to load
[Geben Sie den Namen des Programms ein, das geladen werden soll]
Geben Sie den Namen des Programms ein, das Sie debuggen wollen. Sie können DOS-Jokerzeichen benützen, um eine Auswahlliste von Dateien zu erhalten, oder den Dateinamen direkt eingeben, um das Programm zu laden. Wenn Sie den Dateinamen ohne Erweiterung eingeben, wird automatisch die Erweiterung ».EXE« angehängt.

Enter read file name
[Geben Sie den Namen des Programms ein, das geladen werden soll]
Geben Sie den Namen der Datei ein, die Sie einlesen wollen. Wenn Sie Jokerzeichen benützen oder die Voreinstellung »*.*« verwenden, erhalten Sie eine Auswahlliste mit den verfügbaren Dateien.

Enter search bytes
[Geben Sie die gesuchten Bytes ein]
Geben Sie eine Byte-Folge ein, die ab der Speicherstelle, in der der Cursor steht, gesucht werden soll. In Kapitel 9 »Ausdrücke« finden Sie eine vollständige Beschreibung der Byte-Listen.

Enter search instruction or bytes
[Geben Sie den Befehl oder die Bytes ein, die gesucht werden sollen]
Geben Sie eine Anweisung ein, die so aussieht, als würden Sie auf den Befehl *Assemble* im lokalen Menü antworten, oder geben Sie eine Byte-Folge wie bei einem *Search*-Befehl in einem Datenausschnitt ein.

Enter search string
[Geben Sie den gesuchten String ein]
Geben Sie den String ein, der gesucht werden soll. Sie können einfache Jokerzeichen verwenden, um ähnliche Strings zu suchen. Zum Beispiel kann das Sternchen (*) für null oder mehrere beliebige Zeichen und das Fragezeichen (?) für ein beliebiges Zeichen verwendet werden.

Enter source address, destination, count
[Geben Sie Quell- und Zieladresse sowie die Anzahl ein]
Geben Sie die Adresse des Blocks ein, den Sie kopieren möchten, die Adresse, wohin Sie ihn kopieren wollen, und die Anzahl der Byte, die Sie kopieren wollen. Trennen Sie die drei Ausdrücke durch Kommata voneinander.

Enter source directory path
[Geben Sie die Verzeichnispfade für Quellcodedateien ein]
Geben Sie eine Liste von Verzeichnissen ein, die durch Leerzeichen oder Strichpunkte (;) getrennt sind. Diese Verzeichnisse werden in der Reihenfolge, in der sie in dieser Liste erscheinen, nach den Quellcodedateien durchsucht.

Enter symbol table name
[Geben Sie den Namen der Symboltabelle ein]
Geben Sie den Namen der Symboldatei ein, die geladen werden soll. Üblicherweise haben diese Dateien die Erweiterung ».TDS«. Die Dateierweiterung muß explizit angegeben werden.

Enter value to fill marked block
[Geben Sie einen Wert ein, um den markierten Block auszufüllen]
Tragen Sie in den markierten Block einen Byte-Wert ein.

Enter variable to inspect
[Geben Sie die Variable ein, die untersucht werden soll]
Geben Sie den Namen der Variablen oder des Ausdrucks ein, dessen Inhalt Sie untersuchen wollen. Falls das Dialogfenster mit einem Wert aus einem Fensterausschnitt initialisiert wird, können Sie den Eintrag sofort mit der Eingabetaste [CR] bestätigen oder nach Belieben ändern.

Enter write file name
[Geben Sie den Namen der Datei ein, in die die Informationen geschrieben werden sollen]
Geben Sie den Namen der Datei ein, in die Sie den Speicherblock schreiben wollen.

Overwrite __ ?
[__ überschreiben ?]
Sie haben den Namen einer Datei angegeben, die bereits existiert. Sie können die Datei überschreiben, und damit den bisherigen Inhalt ersetzen, oder den Befehl abbrechen und die bisherige Datei beibehalten.

Overwrite existing macro on selected key?
[Soll das vorhandene Makro für die ausgewählte Tastenkombination überschrieben werden?]
Sie haben eine Tastenkombination gedrückt, um ein Makro aufzunehmen, und dieser Tastenkombination ist schon ein Makro zugeordnet. Wenn Sie das bestehende Makro überschreiben wollen, geben Sie [Y] ein, sonst drücken Sie [N], um den Befehl rückgängig zu machen.

Pick a method name
[Wählen Sie eine Methode aus]
Sie haben den Namen einer Routine angegeben, der sich auf mehr als ein Objekt bezieht. Wählen Sie die gewünschte Routine in der angegebenen Liste aus.

Pick a module
[Wählen Sie ein Modul aus]
Wählen Sie den Namen eines Moduls aus, das im *Module*-Fenster angezeigt werden soll. Sie erhalten eine Liste aller Module in Ihrem Programm. Wenn Sie eine Datei anschauen möchten, die kein Programmodul ist, verwenden Sie den Menübefehl *View/File*.

Pick a name
[Wählen Sie einen Namen aus]
Wählen Sie aus der Liste der dargestellten Symbole einen Namen aus. Sie können auch einen Namen eintippen; Sie landen dann bei dem Listenelement, dessen Namen mit der von Ihnen eingegebenen Buchstabenfolge beginnt.

Pick a source file
[Wählen Sie eine Quellcodedatei aus]
Wählen Sie eine Quellcodedatei aus der Liste auf dem Bildschirm aus. Es werden nur die Quellcodedateien gezeigt, die zum aktuellen Modul gehören.

Pick a window
[Wählen Sie ein Fenster aus]
Wählen Sie ein Fenster aus der Liste mit den Überschriften der aktiven Fenster aus.

Pick macro to delete
[Wählen Sie das Makro aus, das Sie löschen wollen]
Geben Sie die Taste oder die Tastenkombination des Makros ein, das Sie löschen wollen. Die Tasten erhalten dann wieder die Funktion, die sie vor der Zuordnung des Makros hatten.

Press key to assign macro to
[Drücken Sie eine Taste für eine Makrozuordnung]
Drücken Sie die Taste, der Sie das Makro zuordnen wollen. Dann geben Sie die Befehlsfolge ein, die Sie der gewählten Makrotaste zuordnen wollen. Die Befehlsfolge wird tatsächlich ausgeführt, während Sie sie eingeben. Um die Makroaufnahme zu beenden, drücken Sie noch einmal die Taste, der Sie das Makro zugeordnet haben oder betätigen Sie Alt -.

Program already terminated, Reload?
[Programm bereits beendet; neu laden?]
Sie haben versucht, Ihr Programm laufen zu lassen oder einen Befehl auszuführen, nachdem das Programm bereits beendet wurde. Wenn Sie Y eingeben, wird Ihr Programm neu gestartet, wenn Sie mit N antworten, wird Ihr Programm nicht neu gestartet und die Befehle zum Ablaufen des Programms werden nicht ausgeführt.

Reload program so arguments take effect?
[Programm neu laden, damit die Argumente übernommen werden?]
Sie haben gerade die Kommandozeilenargumente für das Programm geändert, das Sie debuggen. Wenn Sie Y eingeben, wird Ihr Programm neu geladen und an den Programmanfang gesetzt. Sie müssen das normalerweise dann machen, nachdem Sie die Argumente geändert haben. Viele Programme, die mit Programmiersprachen von Borland erstellt wurden, überprüfen die Argumente nur einmal: wenn das Programm geladen wird. Alle nachfolgenden Änderungen der Programmargumente werden nicht beachtet, bis das Programm neu gestartet wird.

Die Fehlermeldungen

TDW verwendet Fehlermeldungen, um Ihnen Informationen zu unerwarteten Vorgängen zu geben. Manchmal bedeutet das, daß ein Befehl nicht ausgeführt werden konnte; in anderen Fällen konnten ihre Befehle nicht exakt ausgeführt werden.

Fehlermeldungen werden normalerweise von einem Alarmton begleitet. Sie können den Alarmton mit Hilfe des Installationsprogramms TDWINST abschalten.

Schwerwiegende Fehler

Bei allen schwerwiegenden Fehlern verabschiedet sich TDW und kehrt sofort zu Windows zurück. Manche dieser Fehler treten beim Aufruf des Debuggers auf. Andere passieren dann, wenn etwas Irreparables geschieht, während man mit TDW arbeitet. In beiden Fällen haben Sie keine andere Möglichkeit, als den Debugger nach der Lösung des Problems erneut zu starten.

Bad or missing configuration file
[Konfigurationsdatei ist fehlerhaft oder exisitert nicht]
Die Konfigurationsdatei ist entweder zerstört oder keine Konfigurationsdatei für TDW.

Invalid switch: ___
[Unzulässige Option: ___]
Sie haben beim Aufruf des Debuggers eine unzulässige Option angegeben. Anhang A, »Kommandozeilenoptionen«, enthält eine Aufstellung mit allen Kommandozeilenoptionen. In Kapitel 4, »Der Aufruf von TDW«, werden diese Optionen eingehend erläutert.

Not enough memory
[Zuwenig Speicherplatz frei]
TDW hat nicht mehr genug Speicherplatz, um Ihren Befehl auszuführen.

Old configuration file
[Veraltete Konfigurationsdatei]
Sie haben versucht, TDW mit einer Konfigurationsdatei einer früheren Version zu starten. Sie müssen neue Konfigurationsdateien für die verwendete Version von TDW anlegen.

Unsupported video adapter
[Video-Adapter wird nicht unterstützt]
TDW kann nicht feststellen, welchen Bildschirmadapter Sie verwenden. Es werden nur Adapter des Typs EGA, VGA, Hercules und Super-VGA unterstützt.

Andere Fehlermeldungen

')' expected
[')' erwartet]
Bei der Auswertung eines Ausdrucks wurde festgestellt, daß eine abschließende Klammer fehlt. Dies passiert, wenn ein korrekter Ausdruck mit einer linken Klammer anfängt, aber nicht mit einer passenden rechten Klammer abgeschlossen wird. Zum Beispiel sollte

```
3*(7+4
```

wie folgt lauten:

```
3*(7+4)
```

':' expected
[':' erwartet]
Bei der Auswertung eines C-Ausdrucks wurde ein Fragezeichen (?) gefunden, mit dem die beiden ersten Ausdrücke des dreiwertigen Operators ?: getrennt sind. Es wurde jedoch kein entsprechender Doppelpunkt (:) gefunden, der den zweiten und dritten Ausdruck voneinander getrennt hätte. Zum Beispiel sollte

```
x < 0 ? 4 6
```

wie folgt formuliert werden:

```
x < 0 ? 4 : 6
```

']' expected
['']' erwartet]
Bei der Auswertung eines Ausdrucks wurde festgestellt, daß eine abschließende eckige Klammer fehlt. Dies passiert, wenn ein Array-Index mit einer linken eckigen Klammer anfängt, aber nicht mit einer passenden rechten eckigen Klammer abgeschlossen wird. Zum Beispiel sollte

```
Tabelle [4
```

wohl

```
Tabelle [4]
```

heißen. Dieser Fehler kann auch dann auftreten, wenn ein Assemblerbefehl mittels des integrierten Assemblers eingegeben wurde. In diesem Fall wurde eine linke eckige Klammer bei einem Speicherzugriff mit einem Basis- oder einem Indexregister angetroffen, ohne daß der Ausdruck mit der rechten eckigen Klammer abgeschlossen worden wäre. Zum Beispiel sollte

```
mov    ax,4[si
```

als

```
mov    ax,4[si]
```

geschrieben werden.

Already logging to a file
[Protokoll erfolgt bereits auf Datei]
Sie haben den Befehl *Open log file* angegeben, obwohl derselbe Befehl bereits eingegeben wurde, ohne daß zwischendurch der Befehl *Close log file* erfolgt wäre. Wenn Sie eine andere Protokolldatei verwenden wollen, müssen Sie die aktuelle Protokolldatei vorher mit dem Befehl *Close log file* schließen.

Ambiguous symbol name
[Mehrdeutige Symbolbezeichnung]
Sie haben einen Symbolnamen in einem Ausdruck eingegeben, der sich innerhalb eines Programms in C++ oder objektorientierten Pascal nicht eindeutig zuordnen läßt, und darüber hinaus kein Symbol aus der angebotenen Liste ausgewählt. Sie müssen das entsprechende Symbol aus der dargestellten Liste auswählen, bevor der Ausdruck ausgewertet werden kann.

Bad or missing configuration file name
[Falscher Name der Konfigurationsdatei]
Die hinter der Kommandozeilenoption *-c* angegebene Konfigurationsdatei existiert nicht.

Cannot access an inactive scope
[Auf einen nicht aktivierten Gültigkeitsbereich kann nicht zugegriffen werden]
Sie haben einen Ausdruck eingegeben oder auf eine Variable in einem *Module*-Fenster zugegriffen, die nicht im Bereich der aktiven Funktion liegt. Variablen in einer nicht aktivierten Funktion haben keinen definierten Wert, deshalb können Sie sie nicht in Ausdrücken verwenden oder deren Werte anschauen.

Cannot be changed
[Kann nicht verändert werden]
Sie haben versucht, ein Symbol zu ändern, das nicht verändert werden kann. Die einzigen Symbole, die direkt verändert werden können, sind skalare Daten (`int`, `long`, usw. in C; `byte`, `integer`, `longint` usw. in Pascal) und Zeiger. Wenn Sie eine Struktur oder ein Array verändern wollen, müssen Sie die Elemente einzeln nacheinander verändern.

Can't have more than one segment override.
[Kann nicht mehr als eine Segmentvorgabe verwenden]
Sie wollten einen Befehl assemblieren, in dem beide Operanden eine Segmentvorgabe hatten. Nur einer der beiden Operanden darf jedoch eine Segmentvorgabe besitzen. Zum Beispiel sollte

```
mov    es:[bx],ds:1
```

als

```
mov    es:[bx],1
```

oder

```
mov    ax,[1]
mov    es:[bx],ax
```

geschrieben werden.

Can't set a breakpoint at this location
[Breakpoint kann nicht an dieser Stelle gesetzt werden]
Sie wollten einen Breakpoint im ROM, in einem nicht vorhandenen Speicherbereich oder im Segment 0 setzen. Die einzige Möglichkeit, um die Ausführung eines Programms im ROM nachzuvollziehen, ist der Befehl *Run/Trace Into*, bei dem man Befehl für Befehl beobachten kann.

Can't set any more hardware breakpoints
[Es können keine weiteren Hardware-Breakpoints mehr gesetzt werden]
Sie können keinen weiteren Hardware-Breakpoint setzen, ohne vorher einen alten Breakpoint zu löschen. Unterschiedliche Hardware-Debugger unterstützen unterschiedlich viele und verschiedene Arten von Hardware-Breakpoints.

Can't set hardware condition on this breakpoint
[Hardware-Bedingung kann für diesen Breakpoint nicht gesetzt werden]
Sie wollten eine Hardware-Bedingung für einen Breakpoint angeben, der nicht als globaler Breakpoint vorliegt. Hardware-Bedingungen lassen sich nur für globale Breakpoints angeben.

Can't set that sort of hardware breakpoint.
[Diese Art von Hardware-Breakpoint kann nicht gesetzt werden]
Der Hardware-Gerätetreiber, den Sie in der CONFIG.SYS-Datei installiert haben, unterstützt keinen Hardware-Breakpoint in der von Ihnen angegebenen Kombination aus Bus-Zyklus-Typ, Adreß- und Datenübereinstimmung.

Constructors and destructors cannot be called
[Konstruktoren und Destruktoren können nicht aufgerufen werden]
Diese Fehlermeldung erscheint nur dann, wenn Sie ein Programm debuggen, in dem Objekte verwendet werden. Sie haben wahrscheinlich versucht, eine Objektmethode oder Elementfunktion auszuwerten, die entweder ein Konstruktor oder ein Destruktor ist. Dies ist nicht zulässig.

Count value too large
[Wert für Blocklänge zu hoch]
Sie haben im Datenabschnitt des CPU-Fensters für eines der lokalen Menükommandos eine zu große Blocklänge angegeben. Die Blocklänge darf nicht höher sein als FFFFFh.

Ctrl-Alt-SysRq interrupt. System crash possible. Continue?
[Ctrl-Alt-SysRq Interrupt. Systemabsturz möglich. Weitermachen?]
Sie haben entweder versucht, TDW zu verlassen oder Ihr Anwendungsprogramm nochmals zu laden, während das Programm infolge der Eingabe von Strg Alt S-Abf unterbrochen war. Da der Windows-Kernel während der Zeit, da Sie die Anwendung verlassen hatten, aktiv war, führt das Verlassen von TDW oder das erneute Laden der Anwendung zu unvorhersehbaren Ergebnissen (vermutlich zu einem »Systemabsturz« und damit zu einem Neustart).

Falls möglich, setzen Sie in Ihrem Code einen Breakpoint, der dafür sorgt, daß Ihr Programm zu TDW zurückkehrt und starten das Programm neu. Wenn Ihr Programm dann auf den Breakpoint stößt und die Steuerung an TDW übergeht, können Sie TDW beenden oder das Programm neu laden.

Destination too far away
[Zieladresse zu weit entfernt]
Sie wollten einen bedingten Sprungbefehl assemblieren, bei dem die Zieladresse zu weit von der aktuellen Position entfernt ist. Die Zieladresse eines bedingten Sprungbefehls muß innerhalb eines Bereichs von -128 und +127 Bytes relativ zur aktuellen Position entfernt liegen.

Divide by zero
[Division durch Null]
Sie haben einen Ausdruck mit einem Divisions- (`/`, `div`) oder Modul-Operator (`mod`, `%`) eingegeben, dessen Ausdruck auf der rechten Seite den Wert Null ergab. Da die Operatoren in diesem Fall kein definiertes Ergebnis haben, wurde eine Fehlermeldung ausgegeben.

DLL already in list
[DLL bereits in der Liste]
Sie haben im Dialogfeld *View/Modules* versucht, eine DLL in die Liste der DLLs und Programme aufzunehmen, doch diese DLL ist in der Liste bereits enthalten.

Error opening file ___
[Fehler beim Öffnen der Datei ___]
TDW war nicht in der Lage, die Datei zu öffnen, die Sie im Datei-Fenster anschauen wollen.

Error opening log file ___
[Fehler beim Öffnen der Log-Datei ___]
Die Datei, die Sie im Befehl *Log to file* angegeben haben, konnte nicht geöffnet werden. Entweder ist nicht mehr genug Speicherplatz vorhanden, um die Datei zu öffnen, oder das angegebene Laufwerk, Verzeichnis oder der Name der Datei ist nicht zulässig. Machen Sie entweder etwas Platz, indem Sie einige Dateien auf dem Laufwerk löschen oder geben Sie einen zulässigen Pfad für die Datei an.

Error reading block into memory
[Fehler beim Einlesen eines Speicherblocks]
Der von Ihnen angegebene Block konnte nicht von der Diskette in den Speicher

geladen werden. Sie haben wahrscheinlich mehr Bytes einlesen wollen, als in der Datei vorhanden sind.

Error saving configuration
[Fehler beim Speichern der Konfigurationsdatei]
TDW konnte Ihre Konfiguration nicht auf das Laufwerk schreiben. Vergewissern Sie sich, daß auf Ihrem Laufwerk genug Speicherplatz frei ist.

Error writing block to disk
[Fehler beim Schreiben eines Speicherblocks auf das Laufwerk]
Der angegebene Block konnte nicht in die angegebene Datei geschrieben werden. Wahrscheinlich überschreitet der Block den noch verfügbaren Speicherplatz auf dem Laufwerk.

Error writing log file__
[Fehler beim Schreiben der Protokolldatei]
TDW konnte die Ausgaben im Fenster *Log* nicht in die angegebene Datei schreiben, weil es zu einem Fehler kam. Möglicherweise ist nicht genug Platz auf dem Laufwerk frei.

Error writing to file
[Fehler beim Schreiben in eine Datei]
TDW konnte Ihre Änderungen nicht in die Datei zurückschreiben. Die Datei war möglicherweise schreibgeschützt oder während des Schreibvorgangs ist ein Hardware-Fehler aufgetreten.

Expression too complex
[Ausdruck zu komplex]
Der Ausdruck, den Sie eingegeben haben, ist zu komplex. Sie müssen einen Ausdruck eingeben, der weniger Operatoren und Operanden hat. Sie können bis zu 64 Operatoren und Operanden in einem Ausdruck verwenden. Operanden sind zum Beispiel Konstanten und Namen von Variablen. Operatoren sind zum Beispiel das Pluszeichen (+), der Zuweisungsoperator (`=` oder `:=`), eine Elementauswahl in Strukturen – und eine Mengenzuordnung (`in`).

Expression with side effects not permitted
[Ausdruck mit Seiteneffekten nicht zulässig]
Sie haben einen Ausdruck eingegeben, der bei der Auswertung eine Speicherstelle verändert. Sie können diese Art von Ausdrücken nicht angeben, wenn TDW einen Ausdruck möglicherweise wiederholt auswerten muß, wie zum Beispiel in einem *Inspector*- oder *Watches* -Fenster.

Extra input after expression
[Zusätzliche Zeichen hinter dem Ausdruck]
Sie haben einen Ausdruck eingegeben, der zwar gültig war, aber hinter dem Ausdruck stehen weitere Angaben. Das kann heißen, daß Sie einen Operator in Ihrem Ausdruck vergessen haben. Zum Beispiel kann es sein, daß

```
3 * 4 + 5 2
```

folgendes

```
3 * 4 + 5 / 2
```

heißen sollte. Oder

```
add    ax,4 5
```

sollte

```
add    ax,45
```

heißen. Es kann auch sein, daß Sie bei der Eingabe einer Zahl eine für die verwendete Programmiersprache unverträgliche Syntax verwendet haben (im Assemblermodus etwa 0xF000 statt 0F000h).

Help file ___ not found
[Hilfedatei ___ nicht gefunden]
Sie haben nach Hilfestellungen gefragt, aber die Datei mit den Hilfebildschirmen konnte nicht gefunden werden. Stellen Sie sicher, daß sich die Hilfedatei im selben Verzeichnis wie das Debuggerprogramm befindet.

Immediate Operand out of range
[Konstanter Operand zu groß]
Sie haben eine Anweisung eingegeben, die einen Operanden in Byte-Größe mit einer Konstanten kombiniert, die zu groß für ein Byte ist. Zum Beispiel sollte

```
add    BYTE PTR [bx],300
```

vielleicht

```
add    WORD PTR [bx],300
```

heißen.

Initialization not complete
[Initialisierung nicht vollständig]
Sie haben versucht auf eine Variable in Ihrem Programm zuzugreifen, bevor das Datensegment vom Initialisierungscode des Compilers richtig geladen wurde. Sie müssen den Initialisierungscode des Compilers bis zum Beginn Ihres Quellcodes ausführen, bevor Sie auf die meisten Programmvariablen zugreifen können.

Invalid character constant
[Unzulässige Zeichenkonstante]
Der eingegebene Ausdruck enthält eine falsche Angabe einer Zeichenkonstanten. Eine Zeichenkonstante besteht aus einem einfachen Anführungszeichen (') , hinter dem ein einziges Zeichen steht und das mit einem weiteren einfachen Anführungszeichen abgeschlossen ist. Zum Beispiel:

```
'A = 'a'
```

sollte heißen

```
'A'= 'a'
```

Invalid format string
[Ungültiger Formatstring]
Sie haben eine Formatierungsangabe hinter einem Ausdruck eingegeben. Der Formatierungsstring ist jedoch ungültig. In Kapitel 9, »Ausdrücke«, finden Sie eine Beschreibung der Formatierungsangaben.

Invalid function parameter(s)
[Ungültige Funktionsparameter]
Sie haben in einem Ausdruck versucht, eine Funktion aufzurufen, haben aber nicht die richtigen Parameter zum Funktionsaufruf eingegeben.

Invalid instruction
[Unzulässiger Befehl]
Sie haben einen Befehl zum Assemblieren eingegeben, der eine gültige Befehlskurzform hatte, aber der von Ihnen angegebene Operand ist nicht zulässig. Das passiert meistens beim Versuch, einen POP CS-Befehl zu assemblieren.

Invalid instruction mnemonic
[Unzulässiger Befehl]
Bei der Eingabe eines Befehls, der assembliert werden soll, haben Sie die Befehlskurzform vergessen. Ein Befehl besteht aus einer Befehlskurzform, gefolgt von optionalen Argumenten. Zum Beispiel:

```
AX,123
```

sollte heißen

```
MOV     ax,123
```

Invalid number entered
[Unzulässige Zahleneingabe]
Sie haben in einem *File*- oder *Module*-Fenster eine unzulässige Zahl im *Goto*-Befehl eingegeben.Die Zahlen müssen dezimal und größer als Null sein.

Invalid operand(s)
[Unzulässig(e/r) Operand(en)]
Der Befehl, den Sie assemblieren wollen, enthält einen oder mehrere Operanden, die nicht zugelassen sind. Zum Beispiel kann ein MOV-Befehl nicht zwei Speicheroperanden enthalten und einige Befehle arbeiten nur mit Word-Operanden. Zum Beispiel:

```
POP     al
```

sollte heißen

```
POP     ax
```

Invalid operator/data combination
[Ungültige Kombination aus Daten und Operatoren]
Sie haben einen Ausdruck eingegeben, in dem ein Operator einen Operanden hat, mit dem die angegebene Operation nicht durchgeführt werden kann. Das ist zum Beispiel der Fall, wenn Sie eine Konstante mit der Adresse einer Funktion in Ihrem Programm multiplizieren wollen.

Invalid pass count entered
[Ungültige Vorgabe für Durchlaufzähler]
Sie haben einen Durchlaufzähler für einen Breakpoint eingegeben, der nicht zwischen 1 und 65535 liegt. Sie dürfen für den Durchlaufzähler nicht 0 eingeben. Während Ihr Programm läuft, bedeutet ein Durchlaufzähler von 1, daß der Breakpoint bereits beim ersten Mal ausgelöst wird.

Invalid register
[Ungültiges Register]
Sie haben ein ungültiges Fließkommaregister als Teil eines Befehls eingegeben, der assembliert werden sollte. Ein Fließkommaregister besteht aus den Buchstaben ST, hinter denen optional eine Zahl von 0 bis 7 in Klammern stehen kann, wie zum Beispiel ST(4).

Invalid register combination in address expression
[Unzulässige Registerkombination im Adreßausdruck]
Bei der Eingabe eines Befehls, der assembliert werden sollte, haben Sie einen Operanden angegeben, der keine der zugelassenen Kombinationen von Basis- und Indexregister enthielt. Ein Adreßausdruck kann ein Basisregister, ein Indexregister oder eines von beiden enthalten. Die Basisregister sind BX und BP, die Indexregister sind SI und DI. Hier die gültigen Adreßregisterkombinationen:

```
BX    BX+SI
BP    BP+SI
DI    BX+DI
SI    BP+DI
```

Invalid register in address expression
[Unzulässiges Register im Adreßausdruck]
Sie haben einen Befehl zum Assemblieren eingegeben, in dem Sie ein unzulässiges Register in den Klammern ([ ]) eines Adreßausdrucks angegeben haben. In Adreßausdrücken darf man nur die Register BX, BP, SI und DI verwenden.

Invalid symbol in operand
[Unzulässiges Symbol im Operanden]
Beim Assemblieren eines Befehls, haben Sie einen Operanden mit einem Zeichen eingeleitet, das nicht am Anfang eines Operanden stehen darf, wie zum Beispiel der Doppelpunkt (:).

Invalid type cast
[Unzulässige Syntax bei Datentypvorgabe]
Sie haben einen Ausdruck eingegeben, der eine explizite Typvorgabe enthält, die nicht korrekt ist. Eine korrekte Typvorgabe beginnt in C mit einer linken Klammer, enthält eine möglicherweise zusammengesetzte Deklaration des Datentyps (ohne den Variablennamen) und endet mit einer rechten Klammer. Zum Beispiel:

```
(x *)p
```

sollte heißen

```
(struct x *)p
```

Invalid value entered
[Ungültiger Wert eingegeben]
Als Sie nach einer Speicheradresse gefragt wurden, haben Sie eine Fließkommazahl statt einer Integerzahl eingegeben.

Keyword not a symbol
[Reserviertes Wort darf kein Symbol sein] (Gilt nur für C und Assembler)
Der von Ihnen eingegebene C-Ausdruck enthält ein reserviertes Wort an einer Stelle, an der ein Variablenname erwartet wird. Sie dürfen reservierte Wörter mit Ausnahme von `sizeof` nur in einer Datentypumwandlung verwenden. Zum Beispiel sollte

```
floatzahl = char charWert
```

wohl

```
floatzahl = (char) charWert
```

heißen.

Left side not a record, structure, or union
[Auf der linken Seite steht keine Struktur oder Variante]
Sie haben einen Ausdruck eingegeben, in dem ein C-Strukturselektor (`.` oder `->`) oder ein Pascal-Record-Feldselektor (`.`) steht. Vor diesem Symbol steht aber weder ein Record- oder Strukturname, noch ein Zeiger auf einen Record oder eine Struktur.

No coprocessor or emulator installed
[Kein Koprozessor oder Emulator installiert]
Sie haben das Fenster *Numeric processor* mit dem Befehl *View/Numeric processor* anlegen wollen, aber es ist kein Koprozessor installiert und der Software-Emulator wird ebenfalls nicht verwendet.

No hardware debugging available
[Kein Hardware-Debugger verfügbar]
Sie wollten einen Breakpoint setzen, bei dem man einen Hardware-Debugger benötigt, aber Sie haben keinen Gerätetreiber für einen Hardware-Debugger

installiert. Sie können diese Fehlermeldung auch dann erhalten, wenn Ihr Gerätetreiber die benötigte Hardware nicht findet.

No help for this context
[Keine Hilfeinformationen in diesem Zusammenhang verfügbar]
Sie haben F1 gedrückt, um Hilfe zu erhalten, aber TDW hat keinen entsprechenden Hilfebildschirm. Teilen Sie den Sachverhalt bitte schriftlich der Firma Borland GmbH mit.

No modules with line number information
[Keine Module mit Zeilennummerinformationen]
Sie haben den Befehl *View/Module* verwendet, aber TDW kann keine Module mit Debug-Informationen finden, um diese Module im Quelltext darzustellen. In der Regel erhalten Sie diese Meldung, wenn Sie ein Programm ohne Symboltabelle debuggen. Sehen Sie dazu unter der Fehlermeldung *Program has no symbol table* nach.

No previous search expression
[Kein Suchbefehl vorausgegangen]
Sie wollten den Befehl *Next* in einem lokalen Menü eines Textausschnitts aufrufen, aber Sie haben vorher noch keinen *Search*-Befehl eingegeben. Sie können *Next* nur nach einem *Search*-Befehl verwenden.

No program loaded
[Kein Programm geladen]
Sie wollten einen Befehl eingeben, bei dem ein Programm geladen sein muß. Es kann zum Beispiel keiner der Befehle im *Run*-Menü verwendet werden, solange kein Programm geladen ist. Wählen Sie den Befehl *File/Open* aus, um ein Programm vor der Verwendung dieser Befehle zu laden.

No type information for this symbol
[Keine Datentypinformation für dieses Symbol]
Sie haben einen Ausdruck eingegeben, der den Namen einer Variablen enthält, für die keine Debug-Informationen vorliegen. Das kann passieren, wenn die Variable in einem Modul vorkommt, das ohne die richtigen Debug-Informationen compiliert wurde. Sie können den Datentyp mit einer expliziten Datentypumwandlung im Ausdruck angeben.

Not a function name
[Kein Funktionsname]
Sie haben einen Ausdruck mit einem Funktionsaufruf eingegeben, aber die Bezeichnung vor der linken Klammer, mit der der Funktionsaufruf eingeleitet wird, ist kein Funkionsname. Jedesmal, wenn eine linke Klammer direkt hinter einem Namen steht, nimmt der Parser an, daß es sich um einen Funktionsaufruf handelt.

Not a record, structure, or union member
[Kein Element eines Records, einer Struktur oder Variante]
Sie haben einen Ausdruck mit einem C-Strukturselektor (. oder ->) oder einem Pascal-Record-Selektor (.) angegeben. Hinter dem Symbol steht jedoch kein Element der Struktur oder des Records.

Not enough memory for selected operation
[Nicht genug Speicherplatz für die ausgewählte Operation]
Sie haben einen Befehl eingegeben, bei dem ein Fenster angelegt werden muß, aber es ist nicht mehr genug Speicherplatz für das neue Fenster verfügbar. Sie müssen zuerst einige der anderen Fenster verkleinern oder entfernen, bevor Sie diesen Befehl erneut eingeben können.

Not enough memory to load program
[Zuwenig Speicherplatz, um das Programm zu laden]
Die Symboltabelle Ihres Programms konnte erfolgreich in den Speicher geladen werden, aber es ist nicht mehr genug Speicherplatz für Ihr Programm verfügbar.

Not enough memory to load symbol table
[Zuwenig Speicherplatz, um die Symboltabelle zu laden]
Es ist nicht genügend Speicherplatz verfügbar, um die Symboltabelle in den Speicher zu laden. Die Symboltabelle enthält Informationen, die TDW zur Darstellung des Quelltextes und der Programmvariablen benötigt. Wenn Sie speicherresidente Programme geladen haben, sollten Sie diese entfernen und den Debugger noch einmal starten. Sie können auch versuchen, die Symboltabelle zu verkleinern, indem Sie beim Compilieren nur für diejenigen Module Debuggerinformationen erzeugen, die Sie tatsächlich debuggen wollen.

Wenn diese Meldung erscheint, müssen Sie genug Speicher freigeben, damit damit sowohl das Programm als auch die dazugehörige Symboltabelle Platz finden. Wenn Sie ein bereits geladenes TSR-Programm debuggen wollen, müssen Sie TDW mit der Kommandozeilenoption *-sm* starten, um Speicher für die Symboltabelle des Programms bereitzustellen.

Only one operand size allowed
[Nur eine Operandengröße zulässig]
Sie haben einen Befehl eingegeben, in dem mehr als ein Datentyp angegeben wurde. Sobald Sie die Größe des Operanden bestimmt haben, dürfen Sie sie nicht mehr verändern. Statt

```
mov    WORD PTR BYTE PTR [bx],1
```

hätten Sie besser

```
mov    BYTE PTR [bx],1
```

angeben sollen.

Operand must be memory location
[Operand muß eine Speicherstelle enthalten]
Sie haben einen Ausdruck eingegeben, in dem ein Teilausdruck eine Speicher-

stelle angeben sollte, was aber nicht der Fall war. Zum Beispiel muß man bei den Zuweisungsoperatoren (=, +=, usw.) und den Inkrement- oder Dekrement-Operatoren (++ und --) einen Speicheroperanden auf der linken Seite angeben.

Operand size unknown
[Operandengröße unbekannt]
Sie wollten einen Befehl assemblieren, haben aber die Größe eines Operanden nicht angegeben. Einige Befehle, die mit Bytes oder Worten arbeiten, benötigen eine genaue Angabe der Operandengröße, wenn diese aus den Operanden nicht ersichtlich ist. Statt

```
add    [bx],1
```

hätte man

```
add    BYTE PTR [bx],1
```

verwenden müssen.

Path not found
[Pfad nicht gefunden]
Sie haben ein Laufwerk mit einem Verzeichnis kombiniert, das nicht existiert. Überprüfen Sie, ob Sie das Laufwerk und das Verzeichnis korrekt eingegeben haben.

Path or file not found
[Pfad oder Datei nicht gefunden]
Sie haben einen Dateinamen oder einen Pfad für eine Datei eingegeben, die entweder nicht existiert oder deren Namen falsch eingegeben wurde. Wenn Sie nicht wissen, wie der exakte Name der Datei lautet, die Sie laden wollen, können Sie den Dateinamen mit der Eingabetaste CR aus einer Liste auswählen. Die Bezeichnungen, die in der Liste mit einem Backslash (\) enden, sind Verzeichnisse.

Program has invalid symbol table
[Programm hat ungültige Symboltabelle]
Die Symboltabelle, die am Ende Ihrer Programmdatei steht, ist nicht mehr in Ordnung. Erzeugen Sie eine neue ».EXE«-Datei und laden Sie sie erneut.

Program has no objects or classes
[Programm hat keine Objekte oder Klassen]
Sie haben versucht, in einem Progamm, das nicht objektorientiert ist, ein *View/Hierarchiy*-Fenster zu öffnen.

Program has no symbol table
[Programm hat keine Symboltabelle]
Das Programm, das Sie debuggen wollen, wurde korrekt geladen, aber es enthält keine Symboltabelle. Sie können die Datei immer noch im *CPU*-Fenster debuggen und die nackten Daten untersuchen, aber Sie können sich nicht auf die Symbole in Ihrem Programm beziehen.

Program linked with wrong linker version
[Programm mit falscher Linkerversion gelinkt]
Sie wollten ein Programm debuggen, das veraltete Debuggerinformationen enthält. Linken Sie Ihr Programm mit der letzten Version des Linkers oder compilieren Sie Ihre Datei mit der letzten Version des Compilers nochmal. In Kapitel 4, »Der Aufruf von TDW«, ist beschrieben, wie Sie Ihr Programm zusammen mit den Debug-Informationen compilieren und linken müssen.

Program not found
[Programm nicht gefunden]
Der von Ihnen angegebene Programmname existiert nicht. Geben Sie entweder den korrekten Dateinamen an oder wählen Sie das Programm mit der Auswahlliste aus.

Register cannot be used with this operator
[Register kann nicht mit diesem Operator verwendet werden]
Sie wollten einen Befehl assemblieren, in dem ein Basis- oder Indexregister mit einem negativen Offset verwendet wird. Sie dürfen für die Basis- und Indexregister nur positive Offsets verwenden. Statt

```
INC    WORD PTR [12-BX]
```

darf nur folgender Befehl verwendet werden:

```
INC    WORD PTR [12+BX]
```

Register or displacement expected
[Register oder Offset erwartet]
Sie wollten einen Befehl assemblieren, der einen unzulässigen Ausdruck zwischen den eckigen Klammern enthält. Sie dürfen nur Registerbezeichnungen oder Offset-Angaben in den eckigen Klammern verwenden.

Run out of space for keystroke macros
[Kein Platz mehr für weitere Makros]
Bei der Aufnahme des Makros ist der Platz ausgegangen. Sie können nur bis zu 256 Tastenanschläge für alle Makros speichern.

Search expression not found
[Gesuchter Ausdruck nicht gefunden]
Der gesuchte Text oder die gesuchten Bytes wurden nicht gefunden. Die Suche fängt bei der aktuellen Cursorposition in der Datei an und verläuft nur vorwärts. Wenn Sie eine Datei ganz durchsuchen wollen, müssen Sie vor der Eingabe des Suchbefehls mit Strg Bild↑ an den Anfang der Datei springen.

Source file ___ not found
[Quellcodedatei ___ nicht gefunden]
TDW kann die Quellcodedatei für das Modul, das Sie untersuchen wollen, nicht finden. Vor der Ausgabe dieser Meldung wurde an verschiedenen Stellen gesucht:

- dort, wo der Compiler die Quellcodedatei gefunden hat;
- in den Verzeichnissen, die durch die Kommandozeilenoption *-sd* und den Befehl *Options/Path for source* festgelegt wurden;
- im aktuellen Verzeichnis;
- im Verzeichnis, in dem TDW das Programm gefunden hat, das Sie debuggen.

Sie sollten das Verzeichnis, das die Quellcodedatei enthält, in die Liste der Suchverzeichnisse aufnehmen. Verwenden Sie dazu den Befehl *Options/Path for source*.

Symbol not found
[Symbol nicht gefunden]
Sie haben einen Ausdruck mit einer ungültigen Variablenbezeichnung eingegeben. Sie haben sich vielleicht vertippt oder die Variable befindet sich in einer anderen Prozedur oder Funktion als der aktuellen oder außerhalb des Gültigkeitsbereichs in einem anderen Modul.

Symbol table file not found
[Symboldatei nicht gefunden]
Die angegebene Datei mit der Symboltabelle existiert nicht. Sie können entweder eine ».TDS«- oder ».EXE«-Datei für die Symboltabelle angeben.

Syntax error
[Syntaxfehler]
Sie haben einen Ausdruck mit einem falschen Format angegeben. Dies ist eine allgemeine Fehlermeldung für Fälle, in denen die Fehlerursache nicht näher bestimmt werden kann.

Too many files match wildcard mask
[Zuviele Dateien mit den Jokerzeichen ausgewählt]
Sie haben eine Dateimaske mit Jokerzeichen eingegeben, mit der mehr als 100 Dateien gefunden wurden. Es können aber nicht mehr als 100 Dateien dargestellt werden.

Unexpected end of line
[Vorzeitiges Zeilenende]
Beim Auswerten eines Ausdrucks ist dessen Ende erreicht worden, bevor ein gültiger Ausdruck erkannt werden konnte. Statt

```
99 - 22 *
```

hätte es vielleicht

```
99 - 22 * 4
```

heißen sollen, und

```
SUB    AX,
```

war vielleicht wohl als

```
SUB     AX,4
```

gedacht.

Unknown character
[Unbekanntes Zeichen]
Sie haben einen Ausdruck eingegeben, der ein Zeichen enthält, das in einem Ausdruck nicht verwendet werden darf, wie zum Beispiel das umgekehrte einfache Anführungszeichen (`) in C.

Unknown record, structure, or union name
[Unbekannte Record- oder Strukturbezeichnung]
Sie haben einen Ausdruck eingegeben, der eine Typumwandlung mit einer unbekannten Record-, Struktur- oder Variantenbezeichnung enthält. (In C und Assembler haben Strukturen einen eigenen Symbolraum, den sie nicht mit anderen Variablen teilen müssen.)

Unknown symbol
[Unbekanntes Symbol]
Sie haben einen Ausdruck mit einer ungültigen lokalen Variablen eingegeben. Entweder ist der Modulname falsch oder das Symbol oder die Zeilennummer ist nicht richtig.

Unterminated string
[String ohne Ende]
Sie haben einen String eingegeben, der nicht mit einem Anführungszeichen abgeschlossen wurde (" in C, ' in Pascal). Wenn Sie einen String in Pascal-Syntax eingeben wollen, müssen Sie ein einfaches Anführungszeichen einfügen. Wenn Sie einen C-String eingeben wollen, müssen Sie doppelte Anführungszeichen (") verwenden.

Value must be between nn and nn
[Wert muß zwischen *nn* und *nn* liegen]
Sie haben einen ungültigen Wert eine Editor-Einstellung (etwa für den Tabulatorabstand) eingegeben. Die Fehlermeldung teilt Ihnen die zulässigen Werte mit.

Variable not available
[Variable nicht verfügbar]
Ihr Programmcode wurde optimiert; auf die Variable, die Sie suchen, kann nicht mehr zugegriffen werden.

Video mode not available
[Bildschirmmodus nicht verfügbar]
Sie wollten in den 43-/50-Zeilenmodus umschalten, aber Ihr Bildschirmadapter unterstützt diesen Modus nicht. Sie können den 43-/50-Zeilenmodus nur mit einem EGA-, VGA- oder SVGA-Adapter verwenden.

E. Glossar

Die Ausdrücke, die im Glossar zusammengestellt sind, werden in diesem Handbuch häufig verwendet. Einige davon beziehen sich auf Computer und Software allgemein, andere betreffen Turbo Debugger selbst.

Aktion	Operation, die beim Auslösen eines Breakpoint vorgenommen wird, etwa das Anhalten Ihres Programms, die Auswertung eines Ausdrucks, das Festhalten des Ergebnisses in einem Protokoll.
aktiver Ausschnitt	Alle Benutzereingaben und Befehle für ein lokales Menü beziehen sich auf diesen Ausschnitt.
aktives Fenster	Momentan ausgewähltes Fenster, in dem die Eingaben verarbeitet werden.
Array	Ein Datentyp, der aus einem oder mehreren Objekten desselben Datentyps besteht, die im Speicher hintereinanderliegen und über einen Index adressiert werden können.
ASCII	Standard-Zeichensatz für IBM PC und viele andere Computer.
Assembler	Programm, mit dem man Assembler-Befehle in hexadezimale Maschinenbefehle umwandeln kann.
Ausdruck	Eine Kombination von Operatoren und Operanden, die nach festgelegten Syntaxregeln angeordnet werden müssen.
Ausführung rückwärts	Die Ausführung von Programmschritten ungeschehen machen.
Ausschnitt	Ein Fenster kann in Turbo Debugger aus mehreren Ausschnitten bestehen. Ausschnitte dienen der logischen Gliederung der Informationen. Zum Beispiel weist das *CPU*-Fenster unter anderem einen *Stack*-Ausschnitt und einen Registerausschnitt auf.
Auswahlfeld	Wenn mehrere Einstellungen sich gegenseitig ausschließen, wie z.B. schwarzweiß und Farbe, werden diese Einstellungen in einem Dialogfenster in Auswahlfeldern vorgenommen.

automatische Variable	In C hat eine Variable dieses Typs nur innerhalb des Funktionsblocks Gültigkeit; ihr Wert wird auf dem Stack gespeichert.
Befehlsfeld	Ein Feld eines Dialogfensters, das nach Anklicken mit der Maus oder Aktivierung über die Tastatur sofort einen bestimmten Vorgang auslöst, z.B. Abbruch des Dialogs durch *Cancel*.
Breakpoint	Haltepunkt, an dem das Programm angehalten wird. Der Breakpoint kann bedingt ausgelöst werden, indem man ihn mit Bedingungen und Durchlaufzählern verknüpft.
C-Ausdruck	Ausdruck, wie er in C verwendet werden darf.
CPU	Zentraleinheit des Computers, in PCs die 80x86-Prozessoren. Eine CPU besteht aus Registern, Flags, einer Recheneinheit für logische und arithmetische Operationen, Speicheranschlüssen sowie einer Reihe von Ein-/Ausgabekanälen.
CPU-Flag	Ein Status-Bit der CPU, durch das nachfolgende Operationen beeinflußt werden können, z.B. bedingte Sprungbefehle.
CPU-Register	Schnelle Speicherstelle in der CPU. Die Register sind: AX, BX, CX, DX, SI, DI, BP, SP, IP, CS, DS, ES und SS. Die Extended Register sind: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, PS und GS.
CS:IP	Aktuelle Programmposition, bestehend aus Code-Segment und Programmzeiger.
Datentyp-vorgabe	Ausdrückliche Umwandlung eines Ausdrucks in einen anderen Datentyp, in C zum Beispiel (`char *`).
Dialogfenster	Fenster, in dem der Benutzer seine Eingabe macht. Die letzten Eingaben werden in einer History-Liste gespeichert, mit deren Hilfe man ohne viel Tipparbeit Eingaben wiederholen kann.
Disassembler	Ein Programm, mit dem man Maschinenbefehle in lesbaren Assembler-Code umwandeln kann. In Turbo Debugger werden im Code-Ausschnitt des CPU-Fensters alle Befehle automatisch disassembliert.
EMS	»expanded memory specification« ist eine Art der Speichererweiterung, mit der man mehr als 640 KByte Speicher in einem PC ausnützen kann. Der zusätzliche Speicher ist jedoch nicht direkt adressierbar, sondern muß jedesmal eingeblendet werden, so daß er für Programmcode nur bei einer Overlay-Verwaltung verwendet werden kann. Turbo Debugger nutzt EMS-Speichererweiterungen zum Speichern der Symboltabelle.
Fenster	Abgetrennter Bildschirmbereich, welcher meist einer bestimmten Aufgabe zugeordnet ist. Fenster können überlappend oder nicht überlappend angeordnet werden.

globaler Breakpoint	Breakpoint, der bei jedem einzelnen Befehl ausgelöst werden kann.
Gültigkeitsbereich	Symbole können in einem Programm sehr unterschiedliche Gültigkeitsbereiche haben. Während lokale Variablen nur in der aktuellen Funktion existieren, darf man auf statische Variablen nur innerhalb eines Moduls zugreifen, während auf globale Symbole im gesamten Programm zugegriffen werden kann. Zu jedem Zeitpunkt der Programmausführung ist also ein bestimmter Gültigkeitsbereich gegeben, in dem bestimmte Symbole entweder verfügbar oder nicht verfügbar sind. Mit Hilfe einer ausdrücklichen Bezugsvorgabe kann man bei Turbo Debugger auch auf Symbole zugreifen, die an der aktuellen Programmposition an sich nicht verfügbar wären.
History-Liste	Liste aller vorher erfolgten Eingaben in einem Eingabefeld, die für neue Eingabe übernommen oder modifiziert werden können.
Inspector-Fenster	Fenster zum Untersuchen von Variablen und Datenstrukturen.
Jokerzeichen	Platzhalter für beliebige Zeichen in einem Such-String oder einem DOS-Dateinamen.
Konfigurationsdatei	Datei, in der die Standardeinstellungen für ein Programm gespeichert werden. Dazu zählen zum Beispiel Position, Größe und Farbeinstellung eines Fensters. Jedesmal, wenn das Programm gestartet wird, werden zuerst die Standardeinstellungen aus der Konfigurationsdatei übernommen, die anschließend meist noch mit Hilfe von Kommandozeilenoptionen und auch im Programm selbst verändert werden können.
lokales Menü	Jeder Ausschnitt eines Fensters hat ein lokales Menü mit einer Reihe von Befehlen, mit denen man den Inhalt des jeweiligen Fensters manipulieren kann.
Markierungsfeld	Ein Feld in einem Dialogfenster, mit dem man eine Einstellung ein- oder ausschalten kann.
Menge	Ungeordnete Gruppe von skalaren Elementen. Siehe `enum` in C oder `set` in Pascal.
Menüleiste	In der obersten Zeile des Bildschirms befindet sich eine Menüleiste mit sogenannten Pulldown-Menüs, die bei der Auswahl einer Menüoption erscheinen (Alt und der erste Buchstabe des Menüs). Diese Befehle sind jederzeit verfügbar.
Operand	Datenobjekt, mit dem eine Operation mit Hilfe von Operatoren ausgeführt wird. Im folgenden Beispiel werden die Operanden `x` und `4` in der Speicherstelle `y` addiert: `y = x + 4`

Operator	Meist symbolische Kurzform für eine Operation, wie zum Beispiel die Addition (+) oder die Multiplikation (*).
PATH	Environment-Variable unter DOS, in der alle Verzeichnisse angegeben werden, die bei einem Befehlsaufruf nach der entsprechenden Programmdatei durchsucht werden.
Popup-Menü	Ein Popup-Menü erscheint, wenn es die Situation erfordert. Die lokalen Menüs der Turbo-Debugger-Fenster sind Popup-Menüs.
Pulldown-Menü	Menü, das von einer Menüleiste »heruntergezogen« wird. (Die Menüs in der Menüleiste sind typische PulldownMenüs.)
Remote-Debuggen	Verwendung eines zweiten Systems, auf dem das Programm ausgeführt wird, während auf dem lokalen System der Debugger betrieben wird. Beide Systeme sind über eine serielle Schnittstelle verbunden.
Seiteneffekt	Ein Ausdruck, der den Wert einer Variablen oder einer Speicherstelle ändert.
Selektor	Im Protected-Mode können Speicherbereiche aufgrund der virtuellen Adressierung nicht mehr direkt adressiert werden, sondern müssen über Selektoren angesprochen, die von der Speicherverwaltung des Betriebssystems zur Verfügung gestellt und bestimmten Kategorien zugeordnet werden (Daten oder Code). Außerdem werden für diese Speicherbereiche Zugriffsrechte vergeben (Lesen/Schreiben).
Skalar	Ein grundlegender Datentyp mit abzählbaren Werten, wie `Byte`, `Integer` oder `Boolean` in Pascal oder `char`, `int` oder `long` in C bzw. C++.
Stack	Datenbereich mit einer LIFO-Struktur (*Last In First Out*), auf dem lokale Variablen, Parameter und Adressen von Prozeduren untergebracht werden.
Symbol	Bezeichnung einer Variablen, Konstanten, Prozedur oder Funktion.
Trace	Programm Schritt für Schritt ausführen.
Watches-Fenster	Fenster, in dem Variablen laufend überwacht werden können.

Index

A
Add
  Breakpoints 105
Add comment (Protokoll) 119
Add Watch
  Data 89
Adressen 130
Aktion 101
  festlegen 104
  Glossar 293
Aktionsschalter 21
aktiver Ausschnitt (Glossar) 293
Alt-Funktionstasten 33, 40
Always 63, 110
Animate 77
Arguments
  Run 84
Argumente 3, 78
  ändern 84
Arithmetikprozessor 28, 191
Arrays
  Glossar 293
  untersuchen 93, 94
ASCII (Glossar) 293
ASCII-Dateien 125
Assemble
  CPU-Fenster (Code) 177
Assembler 171, 186
  Befehle 251
  Datentypen 252
  Darstellung 260
  gemischte Darstellung 176
  Glossar 293
  Inline- 251
  Modus 59
  Syntax für Ausdrücke 264
Assembler-Daten
  Arrays untersuchen 97
  Skalare untersuchen 95
  Strukturen untersuchen 97
  Varianten untersuchen 97
  Zeiger untersuchen 96
Assemblieren 177
At
  Breakpoints 103
auflisten
  globaler Heap 120
  lokaler Heap 120
Aufruf 55
  Hilfestellungen 59
Ausdrücke 86, 129
  Auswahl der Programmiersprache 129
  Formatangaben 142
  Glossar 293
  in Assembler 141
  in Breakpoints 103
Ausführung rückwärts (Glossar) 293
Ausgabefenster 30
Ausschnitt
  wechseln 32
  Glossar 293
Auswahl
  sukzessive 25
Auswahlbalken 33
Auswahlfeld (Glossar) 293
automatische Variable (Glossar) 294

B
Back trace 78
Back Tracing 17
Base segment:0 to data
  CPU-Fenster (Daten) 183
Bedingungen 101, 110
Beep on error 263
Befehlsausschnitt 80
  lokales Menü (Execution history) 80
Befehlsfeld (Glossar) 294
Befehlsübersicht 195

Beispielprogramm 43
Benutzerbildschirm 30
Benutzeroberfläche
  Dialogfenster 24
  Fenster 26
  History-Listen 24
  lokale Menüs 22f
  Menüleiste 19
  Tastenkürzel 40
Bereiche untersuchen 98
Bezeichner
  automatische Ergänzung 25
Bildlaufleiste 31
Bildschirm
  Anpassung 258
  Darstellung 60
  direkt in Speicher schreiben 260
  Farbanpassung 258
  Farbeinstellung 256
  Modus 265
  Parameter 258
  »Schnee« 260
  Umschaltung 59, 63, 259
  zweiter 59
Block-Befehle 184
Both (Integer format) 63
Break 110
Breakpoint at 74
Breakpoints 101
  Aktion 101
  alle löschen 103
  Ausdrücke abfragen 103
  Bedingungen 101, 110
  Beispiel 48
  definiert setzen 103
  Durchlaufzähler 113
  entfernen 105
  Fensterbotschaften 239
  für Fensterbotschaften über Fenster-Handle 240
  für Fensterbotschaften über Fensterobjekt 242
  globale 114
  globale Speicherüberwachung 103
  Glossar 295
  Hardware- 103, 110
  hinzufügen 105
  Konzepte 101
  löschen 105
  Menü 102
  mit Bedingungen 115
  mit Durchlaufzähler 115
  normale 113
  Programm bis zum B. laufen lassen 77
  umschalten 102
  untersuchen 105
  View 104
Breakpoints-Fenster 26, 104
  lokales Menü 104, 201
  Übersicht 199
Byte-Folgen 135

C

C
  Ausdrücke 135, 294
  Funktionen aufrufen 139
  Syntax für Ausdrücke 264
C++ 145
  Mehrfachvererbung 29, 146, 147
  objektorientiert 145
Caller
  CPU-Fenster (Code) 175
Change 17
  Beispiel 52
  CPU-Fenster (Daten) 182
  CPU-Fenster (Register) 179
  CPU-Fenster (Stack) 186
  globale Variablen 69
  Inspector 98
  Instanzen 151
  lokale Variablen 70
  Numeric processor (Register) 193
  Watches 91
Changed memory 110
Changed memory global
  Breakpoints 103
Clipboard
  lokale Menüs 206
Close log file 119
Code-Ausschnitt 173
  lokales Menü 175, 201
Control-Ausschnitt
  Numeric processor 193
Control-Funktionstasten 33, 40
Control-Tastenkürzel 263
Copyright 4
CPU-Fenster 28, 172
  Datenausschnitt 181
  Flags (Glossar) 294
  Glossar 294
  lokale Menüs 201
  lokales Menü (Code-Ausschnitt 201
  lokales Menü (Code-Ausschnitt) 175
  lokales Menü (Datenausschnitt 202

lokales Menü (Flagausschnitt) 203
lokales Menü (Selector-Ausschnitt 202
lokales Menü(Registerausschnitt) 203
lokales Menü(Stackausschnitt) 203
Register 252, 294
Registerausschnitt 178
Selector 180
Stack-Ausschnitt 185
Statusausschnitt 178
Create (Options/Macros) 62
CS:IP 177, 294
Cursor-Bewegung 33

D
Data-Menü 86
Übersicht 199
Data/Function return 89
Dateien 121, 125
Fenster 125
Maske 213
Daten
ändern 52
beobachten 49
Strukturen 51
Typen in Assembler 252
Typvorgabe (Glossar) 294
untersuchen 50, 86
Datenausschnitt
CPU-Fenster 181
lokales Menü 182, 202
Zahlendarstellung 181
Debug-Informationen
Turbo C 55
Turbo C++ 55
Debuggen
auf Assembler-Ebene 171
Beispiel 226
Einführung 43
in der Praxis 217
Methodik 15
ObjectWindows-Programm 233
TDODEMOB.EXE 233
Debugger
Einsatzgebiet 16
Beispiel 44
Decimal 63
Decrement
CPU-Fenster (Register) 179
Delete All
Breakpoints 103, 105
Options/Makro 62
Watches 91
Demo-Programme 43
Descend
Inspector 99
Instanzen 151
Dialogfenster 19, 20, 24, 269
akustisches Signal 263
Anpassung 257
Befehle 211
Funktionsweise ändern 263
Glossar 294
Hotkeys 21
im Evaluate-Fenster 87
Mausbedienung 21
und Kommandozeilenoptionen 266
Disassembler
Code-Ausschnitt 174
Glossar 294
Disketten 9
Display as
CPU-Fenster (Daten) 183
File-Fenster 127
Display options (Options) 62
Display swapping 63
DLL
Startup-Code untersuchen 59
DOS Version
ermitteln 72
Dump-Fenster 28, 188, 203
Durchlaufzähler 113

E
Edit
Watches 91
Edit-Menü
Übersicht 198
EGA
43-Zeilenmodus 63, 260
Ein-/Ausgabeports 177
Einführung 43
Eingabefelder 21
Editiermöglichkeiten 211
Einzelschritt 78
Empty
Numeric processor (Register) 193
EMS 72
für Execution history 79
Glossar 294
Ende des Programms 82
Erase log 119
Exception 75
Execute 111
Execute to 77

Execution history 28, 79, 204
  Full history 81
  Reverse Execute 80
Expression true 110
Expression true global
  Breakpoints 103

F
Far code
  CPU-Fenster (Daten) 183
Farbbildschirm 265
Fehlermeldungen 269
  andere 278
  schwerwiegende 277
Fehlermöglichkeiten 219
Fehlersuche 15
Fenster 26
  aktives 30, 293
  Anordnung ändern 33
  Anordnung speichern 33
  Anpassung 256
  Ausgabe- 30
  Bedienung 32
  Beispielprogramme 46
  Bestandteile 30
  Bildlaufleiste 31
  Botschaften protokollieren 243
  Breakpoints 26
  Breakpoints- 104
  CPU 28, 172
  Dump 28, 188
  Execution history 28, 79
  File 27, 125
  Glossar 294
  Hierarchy 29, 145
  Inspector 30
  Inspector- 86, 92
  Konfiguration 34
  Log- 27, 118
  mehrfache 29
  Module 26, 122
  nicht-überlappende 260
  Numeric processor 28, 191
  Rahmen 31
  Register- 28
  Registers 188
  schließen 34
  Schließfeld 32
  Stack 27
  Symbolfeld 32
  Titel 31
  Variables 27, 68
  Vergrößerungsfeld 31
  View-Menü 26
  Watches 26, 90
  wechseln 32
  wiederherstellen 34
  Windows Messages 29
  zoomen 34
  Zoomfeld 32
File/Get Info 72
File-Fenster 27, 125, 127
  lokales Menü 126, 204
File/Open 83
Flag-Ausschnitt
  lokales Menü 203
Follow
  CPU-Fenster (Code) 175
  CPU-Fenster (Daten) 182
  CPU-Fenster (Stack) 185
Formatsteuerung 142
Fragen 5
Full history
  Execution history 81
Function return
  Data 89
Funktionen
  untersuchen 95
Funktionstasten 40, 195
  Control- 40 Alt- 40

G
Get Info 72
Gleitkomma-Emulator 191
Global breakpoint at 75
Globalausschnitt
  lokales Menü 208
globale Breakpoints 114
Go to cursor 76
Goto
  CPU-Fenster (Code) 175
  CPU-Fenster (Daten) 182
  CPU-Fenster (Stack) 185
  File-Fenster 126
  Module-Fenster 125
Groß- und Kleinschreibung 60, 264
Gültigkeitsbereich 131
  Glossar 294
  impliziter 134
  Vorgabe 131

H
Hardware 110
  Breakpoints 103, 110

Debuggen 12
Voraussetzungen 2
Help 39
Hex 63
Dateien 125
Hierachy
Inspector-Fenster 149
Hierarchiebaum 29, 146
Inspect 149
Hierarchie
Instanzen 152
Objektttypen 149
Hierarchy-Fenster 29, 145
Übersicht 207
Hilfestellungen 39
Beispiel 45
in Menüs 41
Menüzeile 40
online 39
System 39
zum Aufruf von TDW 59
History list length 263
History-Liste 24
Länge 263
Glossar 295
Hotkeys 195
benutzerdefinierte 25
Dialogfenster 21
Hotline 5

I
I/O
CPU-Fenster (Code) 177
In byte
CPU-Fenster (Code) 177
Increment
CPU-Fenster (Register) 179
Inline-Assembler 186
Input prompting 263
Inspect 17, 85, 86
Breakpoints 105
Data 86
Execution history 80
globale Variablen 69
Hierarchiebaum 149
Inspector 99
Instanzen 151
Klassen 146
lokale Variablen 70
Module-Fenster 123
Objekttyp/Klasse 146
Objekttypen 146, 149
Stack 72
Watches 91
Inspector-Fenster 30, 69, 85f, 92
Glossar 295
Beispiel 50
Change 98
Daten zurückverfolgen 99
Descend 99
Inspect 99
lokales Menü 98, 209
neue Variablen untersuchen 99
neuer Datentyp 100
neues öffnen 99
New expression 99
Range 98
Type Cast 100
Installation 11
Installationsprogramm 255
Instruction trace 78
Integer-Darstellung 63, 259
Interrupt
Aufrufe bei Programm-aufzeichnung 80
Meldung 75

J
Jokerzeichen 213
Glossar 295

K
Klassen
Hierarchie untersuchen 145
Hierarchiebaum 147
Inspect 146
Inspector-Fenster 148
Instanzen untersuchen 150
Liste aller Typen 146
Mehrfachvererbung 147
Show Inherited 149
Tree 146
Kommandozeilenoptionen 56ff. 249
-? 59, 249
-c 58
-d 59
-do 59, 249
-dp 249
-ds 59, 249
-h 59, 249
-i 249
-l 59, 168, 249
-m 249
-p 60

-r 249
-rp 249
-rs 249
-s 60
-sc 60, 249
-sd 60, 249
Suchverzeichnisse 60
Syntax 249
-v 60
-vb 249
-vg 249
-vn 249
Kommandzeilenaufruf
Beispiele 57
Kommentare in Log-Datei 119
Konfiguration
Fenster 34
Konfigurationsdatei 58, 61
Glossar 295
Konstanten
in Assembler 141
in C 137
Kontext 22
Koprozessor 191
Befehle 252
Control-Ausschnitt 193
Control-Bits 193
Fenster 28, 191
Statusausschnitt 194
Statusbits 194
Register 192

L
Language (Options) 61
LCD-Bildschirm 265
Lieferumfang 9
Line
Module-Fenster 124
Listen 21
Ausschnitte 211
Loaded 74
Locals
Variables 69
Log 111
Log-Datei
Kommentare einfügen 119
öffnen 119
schließen 119
Log-Fenster 27, 118
Länge 262
lokales Menü 119, 204
löschen 119
Lokalausschnitt
lokales Menü 208
lokale Befehle
Übersicht 201
lokale Menüs 22f, 201ff.
Befehle 23
Control-Tastenkürzel 40
Glossar 295
Löschen
Breakpoints 103
Protokoll 119

M
Macros (Options) 62
Delete All 62
Remove 62
Makros 62
beenden 62
Erstellung 25
erzeugen 62
Löschen 62
Markierungsfeld (Glossar) 295
Mausunterstützung 263
Kommandozeilenoptionen 60
Mehrfachvererbung 29, 146f
Meldungen
andere Fehlermeldungen 278
Dialogfenster 269
Fehler- 277
schwerwiegende Fehler- 277
Menge (Glossar) 295
Menüs 75
Anpassung 257
Breakpoints 102
Data 86
Data/Add Watch 89
Data/Evaluate/Modify 86
File (Übersicht) 198
File/Get Info 72
File/Quit 65
lokale 22
lokale Befehle (Übersicht) 201
Mausbedienung 20
Options 61
Options/Display options 62
Options/Environment Tab Size 64
Options/Language 61
Options/Macros / Delete All 62
Options/Path for source 64
Options/Restore options 65
Options/Save options 64
Options/Screen lines 63

Options/Stop Recording 62
Options/Macros 62
Options/Macros/Create 62
Options/Macros/Remove 62
Run/Animate 77
Run/Backtrace 78
Run/Execute to 77
Run/Go to Cursor 76
Run/Run 75
Run/Trace into 76
Run/Until return 77
Run/Instruction trace 78
Run/Program reset 78
Run/Step over 76
Stack-Ausschnitt 185
System (Übersicht) 198
Tastaturbedienung 19
View/Stack 69, 71
View/Breakpoints 104
View/CPU 172
View/File 125
View/Module 122
View/Watches 90
Menübaum 213
Menüleiste 19, 23, 45
Befehle 197
Glossar 295
Methoden
Instanzen 151
untersuchen 151
Mixed
CPU-Fenster 174
CPU-Fenster (Code) 176
Modify TD.EXE 267
Module 3
-Fenster 26, 122f
lokales Menü 123, 205
Modulliste 120
Monochrome-Bildschirm 265
Mouse enabled 263

N

Nachkomme
im Objektbaum 147
Namensergänzung
automatische 25
Near code
CPU-Fenster (Daten) 182
New CS:IP
CPU-Fenster (Code) 177
New expression 99
Instanzen 151
Next
CPU-Fenster (Daten) 182
File-Fenster 127
Module-Fenster 124
No program loaded 74
Numeric Processor 191
Control-Ausschnitt 193
lokale Menüs 206
lokales Menü (Control-Ausschnitt) 206
lokales Menü (Registerausschnitt) 206
lokales Menü (Statusausschnitt) 206
Registerausschnitt 192
Statusausschnitt 194

O

Objekte
Hierarchie untersuchen 145
Instanzen untersuchen 150
Objekthierarchie
Darstellung beim Debuggen 152
objektorientiert
debuggen 233
objektorientiertes C++ 145
Objekttyp-Inspector-Fenster
lokales Menü 149
Objekttype
Hierarchy 149
Objekttypen
Inspect 146, 149
Inspector-Fenster 148
Liste aller 146
Show inherited 149
Tree 146
Offset to data
CPU-Fenster (Daten) 183
Open log file 119
Operand (Glossar) 295
Operatoren
Glossar 296
in Assembler 142
in C 138
Optionen 58
Aufruf von Turbo Debugger 249
Hilfsprogramme 10f
laden 65
speichern 64
Options-Menü 61
Übersicht 200
Origin 72
CPU-Fenster (Code) 175
CPU-Fenster (Stack) 185
Module-Fenster 125

Originaldisketten 9
Out byte
  CPU-Fenster (Code) 177

P
Parameter 58
Parent yes 147
Parser
  Syntax 264
Pascal
  Syntax für Ausdrücke 264
Pass count (Breakpoints/
Set options) 113
PATH (Glossar) 296
Path for source 64
Planungsphase 226
Popup-Menüs 19, 22
  Glossar 296
Ports 177
Previous
  CPU-Fenster (Code) 175
  CPU-Fenster (Daten) 183
  CPU-Fenster (Stack) 185
  Module-Fenster 124
Probleme 5
Program reset 78
Programm
  akuelle Position 177
  Argumente 78, 84
  Aufzeichnung 79
  beenden 44, 65, 82
  Design 226
  Einzelschritt 76
  erneut laden 83
  fehlerhafte Beispiele 226
  laden 83
  ausfhren 75
  Neustart 44
  Quelltext untersuchen 121
  Reset 83
  rückwärts ausführen 67, 78
  schrittweise ausführen 76
  starten 75
  unterbrechen 81, 114
  Vorbereitung zum Debuggen 55
  zurücksetzen 78
Programmiersprache 129, 264
Programmschritte
  in Execution history 80
Programmstatus 72
Protected Mode
  Selector 180
Protokoll 118
  Anzahl der Zeilen 262
  Datei öffnen 119
  Datei schließen 119
  einschalten 119
  von Variablenwerten 116
Prozessoren
  8086-Befehlssatz 251
  8087-Befehlssatz 252
  80186-Befehlssatz 251
  80286-Befehlssatz 251
  80287-Befehlssatz 252
  80386-Befehlssatz 252
  80486-Befehlssatz 252
  Anzeige 173
Pseudovariablen 136
Pulldown-Menü (Glossar) 19, 296

Q
Quellcode
  Darstellung 260
  Optionen 60
  untersuchen 121
  Verzeichnisse 60, 122

R
Range
  Inspector 98
  Instanzen 150
Rangfolge
  Assembler 141
  C 138
Read word
  CPU-Fenster (Code) 177
README 10
Register
  in C 136
  Koprozessor 252
Registerausschnitt
  CPU-Fenster 178
  lokales Menü 178, 203
  Numeric processor 192
Registers 32-bit 179
Registers-Fenster 28, 188
Remote-Debuggen
  Glossar 296
Remove
  Breakpoints 105
  Watches 91
  Options/Macros 62
reservierte Worte in C 140
Restore options 65

Reverse Execute
  Execution history 80
rückwärts
  Programmausführung 78
Run (Menü) 75
  Arguments 84
  Backtrace 78
  Program reset 78
  Übersicht 199

S
Save configuration file 267
Save options 64
Schaltfelder 21
Schließfeld 32
Schwarzweiß-Bildschirm 265
Screen lines 63
Search
  CPU-Fenster (Code) 176
  CPU-Fenster (Daten) 182
  File-Fenster 126
  Module-Fenster 124
Segment:offset to data
  CPU-Fenster (Daten) 183
Seiteneffekte 88, 139
  Glossar 296
Selector-Ausschnitt 180
  lokales Menü 202
Selektor 180
Set options (Breakpoints) 104
Show
  lokale Variablen 71
Show inherited 149
  Instanzen 151
  Klassen 149
  Objekttypen 149
Skalare
  Glossar 296
  untersuchen 92
Smart 63
Software-Voraussetzungen 2
Source debugging 264
Speicherplatz
  Belegung 72
Speicherüberwachung 103, 114
Stack
  Fenster 27, 71
  Glossar 296
  lokale Variablen 72
  lokales Menü 72, 207
  Position 72
  untersuchen 72
Stackausschnitt
  CPU-Fenster 185
  lokales Menü 185, 203
Startbildschirm
  Assembler 59
Statusanzeige 39
Statusausschnitt
  CPU-Fenster 178
  lokales Menü 179
  Numeric processor 194
Statuszeile 45
Step 17
Step (Meldung) 74
Step over 76
Stop recording (Options / Makro) 62
Stopped at 74
Strings
  Befehle 187
  in C 137
Strukturen
  untersuchen 93
Suche 213
  sukzessive 33
  in Dateien 126
  Kommandozeilenoptionen 60
  Verzeichnisse 57, 64, 263
sukzessive Auswahl 25
Symbole
  Glossar 296
  Groß- und Kleinschreibung 264
  Gültigkeitsbereich 131
  in Assembler 141
  in C 136
Symbolfeld 32

T
Tabulator 64
  Breite 260
Taschenrechnerfunktion 88, 129
Tastaturbefehle 39
Tastenkürzel
  benutzerdefinierte 25
TDCONFIG.TDW 58, 61
TDDEMO 43, 44
TDODEMOB
  Breakpoints für Fensterbotschaften 239
  debuggen 233
  Eigenschaften 233
  Farbzuordnungsfehler 239
  Fehler beim Bildschirmlöschen 246
  Fensterbereichsfehler 243
  Programm erstellen 236

ScribbleWindow definieren 234
UAE-Fehler 237
TDW 55
andere Fehlermeldungen 278
Anpassung 255
Aufruf 55
Fehlermeldungen 277
Grenzen 18
Kommandozeilenoptionen 56, 58
Konfigurationsdatei 61
Optionen laden und speichern 64f
Programmende 65
schwerwiegende Fehlermeldungen 277
TDWINST 255
43/50 Line Mode Startup 260
Aufruf 256
Bildschirmfarben 256
Bildschirmparameter 258
Display swapping 259
Fast screen update 260
Ignore symbol case 264
Integer format 259
Kommandozeilenoptionen und Dialogfenster 266
Konfiguration speichern 267
Language 264
Log list length 262
Maximum tiled watch 260
Optionen 262
Options 263
Permit 43/50 Lines 261
Quit 268
Startup Display 260
Tab size 260
User screen updating 261
verlassen 268
Terminated, exit code 74
Textausschnitte 210
Toggle
Breakpoints 102
CPU-Fenster (Status) 179
Numeric processor (Control-Bits) 194
Numeric processor (Status-Bits) 194
Trace 17, 76
bei Programmaufzeichnung 80
Glossar 296
Meldung 74
Trace into 76
Tracepoint 101
Tree
Klassen 146
Objekttypen 146
Turbo C
Array untersuchen 94
Beispielprogramm 47
Datenstrukturen 51
Debug-Informationen 55
Funktionen untersuchen 95
Initialisierung automatischer Variablen 221
Makros mit Seiteneffekten 223
Skalare untersuchen 92
Strukturen untersuchen 93
Syntax in Ausdrücken 135
überflüssige Strichpunkte 223
Varianten untersuchen 93
Verlust signifikanter Bits 222
Vorzeichenerweiterung 222
Zeiger untersuchen 93
Zeiger-Arithmetik 222
Zuweisung 221
Turbo C++
Debug-Informationen 55
Initialisierung automatischer Variablen 221
Makros mit Seiteneffekten 223
überflüssige Strichpunkte 223
Verlust signifikanter Bits 222
Vorzeichenerweiterung 222
Zeiger-Arithmetik 222
Zuweisung 221
Turbo C++ für Windows
Fehlermöglichkeiten 220
Turbo Debugger 9
Aufruf 249
Disketten 9
Installation 11
Menüs 213
Optionen 249
Type Cast 100
Instanzen 151

U
Unterbrechung
des Programms 81
Until return 77

V
Variablen 68
globale ändern 69
globale überwachen 70
lokale ändern 70
lokale anzeigen 71
lokale überwachen 70

lokale untersuchen 70
protokollieren 116
Variables-Fenster 27, 68
globaler Ausschnitt 69
lokaler Ausschnitt 69
lokales Menü (Global-ausschnitt) 69, 208
lokales Menü (Lokalausschnitt) 208
statischer Ausschnitt (lokales Menü) 69f
Vererbung
Darstellung beim Debuggen 149
Vergrößerungsfeld 31
Verzeichnisse
der Quelldateien 263
Quellcode 122
Quelldateien 64
Turbo- 263
VGA
50-Zeilenmodus 63, 260
View 17
CPU 172
Fenster 26
Hierarchy 145
Stack 71
Übersicht 198
View Watches 90
Vorfahr im Objektbaum 147

W
Warenzeichen 5
Watch 17
globale Variablen 70
lokale Variablen 70
Module-Fenster 123
Watches 91
Watches
Change 91
Delete All 91
Edit 91
Inspect 91
Remove 91
View 90
Watches-Fenster 26, 90
Glossar 296
Lokales Menü 91, 208
Mausbedienung 49
Watchpoint 101
Werte ändern 98
Window message breakpoint at 74
Windows
Hardware-Debuggen 12
Rückkehr zu 65
Windows Messages
Fenster 29
lokale Menüs 205
lokales Menü (Ausschnitt Message Class) 205
lokales Menü (Ausschnitt Messages) 206
lokales Menü (Ausschnitt Windows Selection) 205
Windows-Anwendungen
Aktion 158
Allgemeines 153
Botschaftenklasse 158, 159, 160
DLL 164
DLL-Liste 165
DLL-Startup-Code 167
Fensterauswahl 154
Heap, global 161
Heap, lokaler 162
Message Class 158
Modullisten 161
Modultypen 163
Nachrichten 154
PDB 162
Programmliste 165
Prozeßdeskriptorblock 162
Set Message Filter 158
Speicherlisten 161
Speicherverwaltungsattribute 162
Startup-Code 167
Typumwandlung 169
Windows-Info-Fenster
anzeigen 120
Write word
CPU-Fenster (Code) 178

Z
Zahlendarstellung 181
Zahlenformat
in C 137
Zeiger
untersuchen 93
zurückverfolgen 99
Zeilennummer
Module-Fenster 47
Zero
CPU-Fenster (Register) 179
Numeric processor (Register) 193
Zoomfeld 32

BENUTZER-
HANDBUCH

TURBO DEBUGGER®

BORLAND

BORLAND GMBH, GAUTINGER STR. 10,POSTFACH 1209, D-8130 STARNBERG, TELEFON 0 81 51 / 263-0, TELEFAX 0 81 51 / 142 09
CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001. (408) 438-530
OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN AND THE UNITED KINDOM * ARTIKEL-NR. 147308